AFRL-IF-RS-TR-2005-268
Final Technical Report
July 2005

WIDELink: A BOOTSTRAPPING APPROACH TO IDENTIFYING, MODELING AND LINKING ON-LINE DATA SOURCES

University of Southern California at Marina Del Ray

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. L835

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2005-268 has been reviewed and is approved for publication

APPROVED: /s/

WILLIAM E. RZEPKA
Project Engineer

FOR THE DIRECTOR: /s/

JOSEPH CAMERA, Chief
Information & Intelligence Exploitation Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	JULY 2005	Final Oct 01 – Mar 05

4. TITLE AND SUBTITLE
WIDELink: A BOOTSTRAPPING APPROACH TO IDENTIFYING, MODELING AND LINKING ON-LINE DATA SOURCES

5. FUNDING NUMBERS
C - F30602-01-C-0197
PE - 31011G
PR - EELD
TA - 01
WU - 02

6. AUTHOR(S)
Craig A. Knoblock, Steven Minton, Kristina Lerman, and Cenk Gazen

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Southern California
Information Science Institute
4676 Admiralty Way
Marina Del Rey California 90292-6695

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington Virginia 22203-1714

AFRL/IFED
525 Brooks Road
Rome New York 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2005-268

11. SUPPLEMENTARY NOTES
AFRL Project Engineer: William E. Rzepka/IFED/(315) 330-2762/ William.Rzepka@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***
A link discovery system must be able to augment its knowledge base by collecting information from diverse, distributed sources. We have developed a system, WideLink, that can automatically extract data from online sources, integrate it into a domain model by automatically labeling it and automatically link it with facts already stored in a knowledge base. The challenge is to locate, extract, and integrate the data that comes from online sources. We addressed these problems by using a bootstrapping approach where the system leverages previously-gathered data, as well as the underlying structure many online data sources have, in order to identify and incorporate new data sources. WideLink systematically explores the structure of online sites so that it is able to retrieve pages on demand from complex web sites (e.g., sites with forms, embedded navigational structures, etc.). The system uses knowledge derived from previously gathered examples to help analyze new types of pages. Using examples of the type of information it is looking for, and characteristic patterns learned from those examples, WideLink can recognize relevant data from new sources, assign it to semantic categories within the domain model, and link it with previously learned facts.

14. SUBJECT TERMS
Information Agents, Information Integration, Web Wrappers, Record Linkage, Semantic Labeling, AgentBuilder

15. NUMBER OF PAGES
62

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Contents

List of Figures

List of Tables

1 Abstract

The goal of the Evidence Extraction (EE) and Link Detection (LD) program was to develop algorithms and techniques to detect connections between people, organizations, places, and things from the masses of data. Much of this data came from the tremendous — and growing — amount of information available online. Given the diverse types of data sources needed for link discovery, it is impractical to manually locate and integrate all of the available information. Thus the ability to automatically or semi-automatically identify, model, and integrate these online resources is a critical capability.

Our goal was to develop a system that augments a knowledge base by collecting information from the diverse data sources, specifically online sources. The system, WIDELink, can **automatically extract** data from online sources, integrate the data into a domain model by **automatically labeling** them and **automatically link** these data with facts already stored in a knowledge base.

Our approach exploits the fact that many online sources that would be useful for link discovery have significant underlying structure since the data often comes from either a database or program. The challenge is to locate, extract, and integrate the data that comes from these sources. We addressed these problems by using a bootstrapping approach where the system leverages previously-gathered information to identify and incorporate new data sources. The WideLink system systematically explores the structure of Web sites so that, unlike most search engines, it is able to retrieve pages on demand from complex web sites (e.g., sites with forms, embedded navigational structures, etc.). Using examples of the type of information it is looking for, and characteristic patterns learned from those examples [25, 26], the system is able to recognize relevant data and assign it to semantic categories within the domain model. Again, bootstrapping makes this possible because the system can use knowledge derived from previously gathered examples to help analyze new types of pages [23]. Finally, the extracted information must be linked with previously gathered facts. We perform this task by building on our previous work on record linkage, which learns mappings between different sources through the use of active learning techniques [38]. These learning techniques determine the importance of various attributes in matching entities. However, these attributes alone may not be sufficient to determine the matches. Therefore, we have also developed techniques to exploit secondary sources to automatically improve matches [27].

2 Overview

The goal of the Evidence Extraction (EE) and Link Detection (LD) program was to develop techniques that detect connections between people, organizations, places, and things from the masses of available data and analyze these connections to detect patterns of suspicious activity. Obviously, performance of LD components can be improved by increasing the amount of relevant data made available to them for analysis. The USC/Fetch partnership has developed the technology for accurately and reliably extracting and integrating data from semi-structured sources, such as lists and tables found on HTML pages. This technology allowed EE and LD components to exploit the wealth of information available online. Our tools use machine learning algorithms 1) to induce wrappers from user-labeled Web pages and 2) to learn rules for linking and consolidating objects across different sources from user-labeled examples. However, the requirement for the user to label relevant data to enable the learning algorithms to work has hampered the ability to use online information in an effective and timely manner. To address this problem, our research has focused on automatically extracting, modeling and linking the information available in online sources. Over the course of the program, the USC/Fetch collaboration has made significant progress toward this goal, as summarized below.

2.1 Automatic Extraction

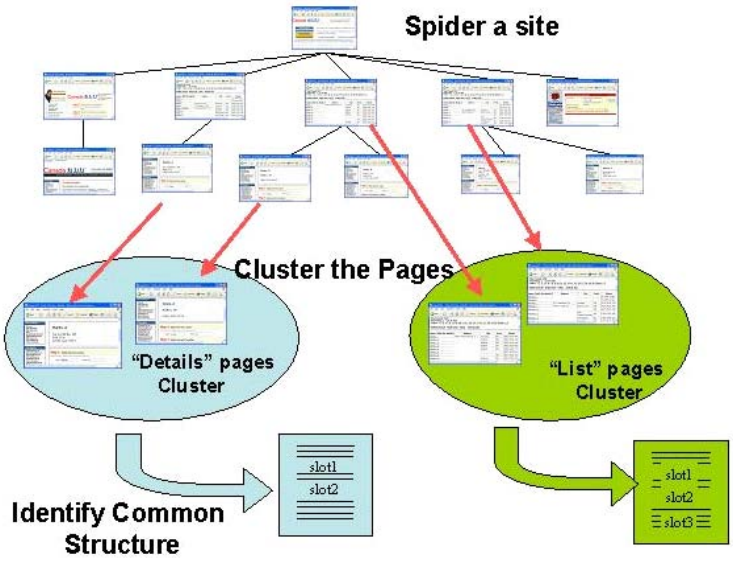

Figure 1: Architecture of the automatic spidering and data extraction system

A large fraction of information on the Web does not exist in static pages, but

rather in proprietary databases. These sites generate the Web pages automatically to display results of user queries. The structure of such sites is surprisingly uniform: they contain a starting or entry page, that allows the user to query or browse the hidden data, results pages containing lists of records retrieved in response to the query, and detail pages that contain additional information about each record. The list pages will look similar to one another, because the same script typically generates them. The same is true of all the detail pages. The information that is common to pages is part of the template, while the information that user is interested in extracting will be in slots, as shown in Figure 1. The USC/Fetch AutoWrap algorithm exploits the site and page structure in order to automatically extract data from the site. AutoWrap starts by spidering the Web site to find all pages that it contains. It then attempts to cluster pages into list and detail classes, as shown in Figure 1. If we knew what class each page belonged to, we could easily deduce the structure, or template, of each class of pages. Likewise, if we knew the structure, we could figure out what class each page belongs to. Therefore, an iterative approach, which clusters the pages, proposes a page template and checks how well it explains the pages, appears to be a solution to this problem. AutoWrap defines a good template according to the Minimum Description Length (MDL) principle: a good template is one that explains the structure of many pages from the site, as well as much of the information contained in these pages. Similarly, a good clustering is one that produces good templates. After AutoWrap clusters pages from the site and finds a template for each class, it uses this template to extract information from the pages (see Section 3.1 for more details). Data that appear on a page in distinct layout elements are extracted as separate tables. In addition, related information is grouped into separate columns in each table. Our unsupervised learning algorithms are able to exploit structure of Web sites to find record boundaries in order to automatically segment HTML tables into individual records (see Section 3.2 for more details).

2.2 Automatic Labeling

The job of the semantic labeling component is to identify known data types among the automatically extracted information. This is done in two steps (see Figure 2): first, we learn the structure of data fields from labeled examples from other sites in the same domain (e.g., coming from existing wrappers), then we apply these patterns to label extracted data. We represent the structure of data by a pattern of tokens and token types. In previous work, we have developed a flexible pattern language and presented an efficient algorithm for learning patterns from examples of a field [26]. Although at present the pattern language contains only specific tokens and syntactic types (such as numeric, capitalized, etc.), we can extend it to include domain-specific semantic types. The algorithm, DataPro, finds patterns that describe many of the examples of a field and are highly unlikely to describe a random token sequence. As an example (see Figure 11b), names can be represented as a set of patterns such as "capitalized word followed by an initial" and "capitalized word followed by

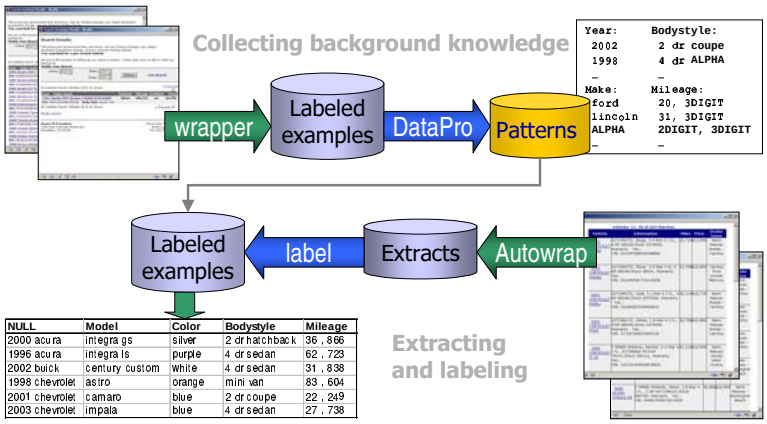

Figure 2: Architecture of the semantic labeling system

a capitalized word," whereas addresses can be represented as "number followed by two capitalized words followed by the word Blvd," etc. Once the patterns for data fields are learned, they can be used to identify and label newly extracted data. Our approach is to check how well the patterns for a field match data in a given column, with a bias towards more specific patterns. Once the data is labeled, it can be provided to the wrapper induction system, along with the new pages, to learn the wrapper for the new site. Section 4 provides more details.

2.3 Automatic Linking

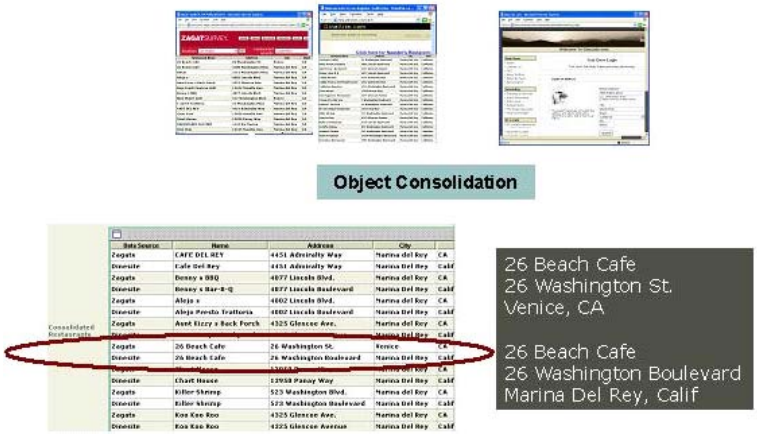

Figure 3: Architecture of the semantic labeling system

Entity consolidation is a challenge faced while integrating data extracted

from multiple Web sites because there may exist inconsistencies among different sources. For example, querying Zagats online guide for a list of local restaurants will return "26 Beach Caf" with an address "26 Washington St. Venice, CA", while querying DineSite.com will return "26 Beach Caf" with an address "26 Washington Blvd. Marina del Rey, CA." Although the street and city names are different, in fact, the two entries refer to the same restaurant (both addresses are correct: Marina del Rey and Venice are adjacent neighborhoods).

USC/Fetch has developed active learning techniques to determine the importance of various attributes in matching entities. However, these attributes alone may not be sufficient to determine the matches. Therefore, we have also developed techniques to exploit secondary sources to automatically improve matches. For example, in the restaurant example in Figure 3, we may supply the restaurant address to an online geocoder, which returns geographic coordinates of the address. Using this information, we determined that the two entries in the example refer to the same restaurant, since they have the same coordinates (see Section 5 for more details).

2.4 Performance Evaluation

We validated our approach on the site incorporation task, which involves both wrapper generation and entity consolidation. Three distinct Web sites containing disparate types of information were used in the evaluation. Evaluations were done at 6 month intervals, starting in January 2002. In Evaluation 1, the wrapper generation and entity consolidation were done manually. Table 1 shows the total time (including user training) required to construct wrappers for these three sites. In Evaluation 2, the entity consolidation was done automatically, although the time cited in the results includes user training (the actual execution time of automatic algorithms is negligible).

In Evaluation 3 we were only able to test the automatic data extraction and labeling part of the automatic wrapper generation process. Currently, these tools are not integrated with the wrapper induction system; consequently, we were unable to generate wrappers. Gain in time due to automating steps of the wrapper generation process comes at a cost, namely accuracy of the results. Table 1 shows accuracy results for different steps of the process. Our continuing goal is to increase robustness and accuracy of the automatic algorithms.

In the following sections we describe in greater detail the three technology components listed above.

3 Automatic Extraction

The World Wide Web is a vast repository of information. The amount of data stored in electronic databases accessible to users through search forms and dynamically generated Web pages, the so-called *hidden Web* [36], dwarfs the amount of information available on static Web pages. Unfortunately, most of this information is presented in a form accessible only to a human user, e.g., ,

Evaluation 1	**time**	**accuracy**
Wrapper Generation	8 hrs 5 min	100%
Entity Consolidation	4 hrs	38%
Total Time	12 hrs 5 min	
Average site incorporation time	4 hrs	
Evaluation 2		
Wrapper Generation	8 hrs 5 min	100%
Entity Consolidation	45 min	73%
Total Time	8 hrs 50 min	
Average site incorporation time	2 hrs 54 min	
Evaluation 3		
Data Extraction and Labeling	32 min	70% extraction 83% labeling
Entity Consolidation	45 min	73%
Total time	1 hr 17 min	

Table 1: Performance results

list or tables that visually lay out relational data. Although newer technologies, such as XML and the Semantic Web, address this problem directly, only a small fraction of the information on the Web is semantically labeled. The overwhelming majority of the available data has to be accessed in other ways.

Web wrappers [31, 21, 15, 10] are popular tools for efficiently extracting information from Web pages. Much of the research in this area over the last decade has been concerned with quick and robust construction of Web wrappers [20], usually with the help of machine learning techniques. Because even the most advanced of such systems learn correct wrappers from examples provided by the user, the focus recently has been on minimizing the number of examples the user has to label, e.g., , through active learning [30]. Still, even when user effort is significantly reduced, the amount and the rate of growth of information on the Web will quickly overwhelm user resources.

There have been numerous attempts at getting applications to make more use out of the information on the web. These attempts can be broadly grouped into extraction and labeling attempts. The extraction work focuses on learning rules for extracting small segments of text. Because this approach requires extra human effort to prepare training sets, it is limited to converting only selected pieces of information to machine-processable form. In the labeling approach, the data is labeled with agreed-upon tags. Labeling also requires extra human effort for maintaining a common set of tags, assigning tags to data and keeping the tags attached to data. Either approach is limited in its applicability because of the extra effort needed.

We have explored a number of new approaches to allow automatic conversion of web-sites into relational form. In the relational form, the data is organized into tables each row of which contains a single entity. Entities may be linked to related entities through references.

Fortunately, dynamically generated Web pages contain much explicit and implicit structure, *both in layout and content*, that we can exploit for purposes of automatic information extraction. The entry point to the Web site is an HTML form for the user to input her query. The result of the query is a list of items or a collection of records from a database. The results are usually displayed as a list or a table on an automatically generated page. We call such a results page the *list page*. Each item or record often has a link to a *detail page* that contains additional information about that entry. Detail pages are also generated automatically and populated with results of database queries. Some of the item attributes are likely to appear on the list page as well as on the detail page.

As others have noted [12], there is important structure in the layout of the list page: some HTML tags are used to separate records, and other HTML tags or text symbols are used to separate columns. In addition to similarities in layout, we expect similarities in the content of data: data in the same column should be of the same type, name or phone number for instance. Of course, the underlying structure of real world data may not be immediately clear: an item may have missing columns; column separators may be different, depending on whether the attribute value is present or missing, or attribute values may be formatted in different ways; a data field may have a lot of variability that we are not able to capture if we assume a uniform layout for each row.

We have developed methods that allow us to efficiently segment data on the list page into records using information contained in detail pages. We describe two approaches to automatic record segmentation: one formulates the task as a constraint satisfaction problem (CSP) and the other uses a probabilistic inference approach. Both techniques exploit the additional information provided by the overlap in the content between list and detail pages. The idea we are trying to leverage is the fact that each detail page represents a separate record. The constraint satisfaction-based technique encodes the relations between data found on the list and detail pages as logical constraints. Solving a constraint satisfaction problem yields a record segmentation. In the probabilistic approach, the information provided by detail pages is used to learn parameters of a probabilistic model. Record segmentation is the assignment of attributes to records that maximizes the likelihood of the data given the model. In addition to computing the record segmentation, the probabilistic approach produces a mapping of data to columns.

3.1 AutoWrap Approach

Automatic extraction and labeling work within the web domain have had varying amounts of success. Most of these approaches focus on a particular structure of web pages, typically the syntactic structure or the link structure. We have built a unifying framework where the multiple types of substructure that are identified by multiple approaches can be combined. The new approach finds a relational representation of a web site by integrating the output of a number of experts each of which look for a particular type of structure.

Our approach relies on the observation that different types of substructure are representative of the underlying structure over different pages of a web site and even over different parts of a single page. If the different substructures can be combined effectively, the resulting representation will be better than any other that uses only one substructure.

The relational form serves as the unifying representation because first, within it, it is possible to represent explicitly the types of substructure that are typical of data on the web and second, it is simple enough that it can be used easily by other agents.

Let's look at some examples of substructure. The following is a list of some of the substructures that can be observed on the single page.

- **Vertical Lists.** There are several locations on the page where the boundaries of items are aligned vertically. This is usually a good indication of an underlying list structure. For example, on the left side of the page, the navigation bar contains a list of links to the departments of the store. In the middle of the page is the list of products, which is the main content of this page. However, not all vertically aligned elements are lists. The entry for a product has several lines (name, item number, link to details, checkbox) which are not items in a list.

- **Horizontal Lists.** Similar to but less common than vertical lists are horizontal lists. Some examples on the page are the navigation bar on the top (Brands, Design Tools, ...) and at the bottom (Corporate, Help Desk, ...). The elements of the latter are vertical lists.

- **Common Formatting.** Visual cues are commonly used to show the structure of the data. For example, the item and model numbers are set in small gray letters whereas the prices are set in large, black, and bold letters indicating that the values in these fields are related to others in the same field but different from other pieces of text on the page.

- **Common Field Features.** The values in a single field usually follow a specific pattern. On this page, the item numbers are five or six digit integers. Model numbers among products of the same brand also have common patterns. Like common formatting, this hints at a relation between the values of the same field.

- **Repeated Values.** The URL attached to the name of a product and that attached to the "View Details..." link are the same for a given product. This structure is valuable in finding the boundaries of an item correctly. The two links and anything in between them are likely to be information about the same item. In addition, the URL for the "Buy Now!" button shares the product id (a superstring of the displayed item number) with the "View Details..." URL.

- **Numbered Lists.** At the bottom of the page, the links to all the pages that contain parts of the product list are indexed by the page numbers (1

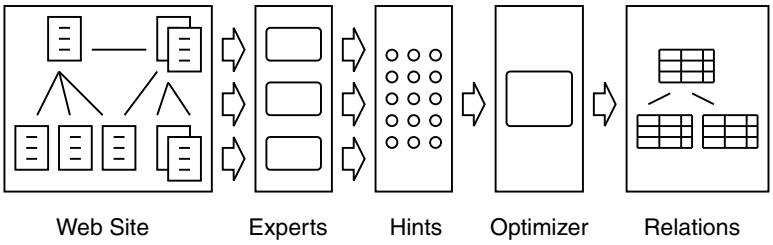

Figure 4: Architecture

and 2). Such sequences of integers are good indicators of list structure. But also note the counter-example of "Page 1 of 2".

Let us now look at some examples of substructure within a collection of pages:

- **Common Page Structure.** The navigation bars at the top, left and bottom of the pages are identical. Also, among the product pages, there is even more common structure. For example, below the product image there is a zoom button, and below the button is a list of product attributes (some of which are common among multiple products). To the right is a list of features for each product, and at the bottom is an optional image showing the customer ratings.
- **Common Content.** Some of the content that is on the navigation page is also on the product pages. For example, product name, item number, model, price and "Buy Now!" URL. This is useful in associating the link on the navigation page with the text around it correctly.
- **Back Link.** On product pages, the URL attached to the "View All Products in this Category" link is the same as the URL of the product list page. This usually indicates a hierarchical structure of the pages rather than just a navigational shortcut.
- **Next Link.** Links that include "next" as a substring of their labels almost always connect to continuation of pages that contain lists.

Our approach (Figure 4) exploits the heterogenous set of substructures by defining a *hint language* in which the the expert responsible for each type of substructure expresses its findings as hints to the optimizer. The best relational representation with respect to the hints is then found by an optimization algorithm.

One piece of information that is missing from the relational representation is column labels (zip code, price, lastname, etc.). This is because the proposed approach focuses on finding the inherent structure of data whereas column labels relate the data to an external, larger knowledge base. However, the relational representation will make it significantly easier for other intelligent agents to label the data.

3.1.1 Page Templates

One of the most important types of substructure arises from the fact that an efficient way to exchange information after similar information has been exchanged is to take the structure of the first exchange and update it with the new parts. Having learned where the information about the weather conditions are in the following segment:

```
<TD VALIGN=MIDDLE ALIGN=CENTER CLASS=obsInfo2 WIDTH=50%>
  <B CLASS=obsTempTextA>54&deg;F</B>
</TD></TR> <TR><TD VALIGN=TOP ALIGN=CENTER CLASS=obsInfo2>
  <B CLASS=obsTextA>Partly Cloudy</B>
</TD> <TD VALIGN=TOP ALIGN=CENTER CLASS=obsInfo2>
  <B CLASS=obsTextA>Feels Like<BR>54&deg;F</B>
</TD></TR>
```

it is easy to write or read this segment:

```
<TD VALIGN=MIDDLE ALIGN=CENTER CLASS=obsInfo2 WIDTH=50%>
  <B CLASS=obsTempTextA>77&deg;F</B>
</TD></TR> <TR><TD VALIGN=TOP ALIGN=CENTER CLASS=obsInfo2>
  <B CLASS=obsTextA>Fair</B>
</TD> <TD VALIGN=TOP ALIGN=CENTER CLASS=obsInfo2>
  <B CLASS=obsTextA>Feels Like<BR>75&deg;F</B>
</TD></TR>
```

This idea of preserving structure as much as possible leads to templates. A *template* is a sequence of alternating *slots* and *stripes* where the stripes are the common strings among all the pages and slots are the place holders for pieces of data that go in between the stripes. A template for the two segments above is shown below. The slots of the template are the labeled boxes.

```
<TD VALIGN=MIDDLE ALIGN=CENTER CLASS=obsInfo2 WIDTH=50%>
  <B CLASS=obsTempTextA>[ actualTemperature ]&deg;F</B>
</TD></TR> <TR><TD VALIGN=TOP ALIGN=CENTER CLASS=obsInfo2>
  <B CLASS=obsTextA>[ currentCondition ]</B>
</TD> <TD VALIGN=TOP ALIGN=CENTER CLASS=obsInfo2>
  <B CLASS=obsTextA>Feels Like<BR>[ feelsLikeTemperature ]&deg;F</B>
</TD></TR>
```

Pages generated using a single template are usually pages describing the details of a single item. Some examples of such pages are pages returned in response to an ISBN query on a bookstore site, a reverse phone-number lookup on white-pages, or a course number query on a college registrar site. When pages are generated from a single template, we will say that they are of the same *page-type*.

One way to find the template of a set of pages is to find the longest common subsequence (LCS) of the token sequences of all the pages. A subsequence of a sequence is the same sequence with some of the elements left out. A sequence is a common subsequence of a set of sequences if it is a subseqeunce of all the

sequences in the set. The LCS, which is not necessarily unique, is a common subsequence that has the maximum length. The LCS immediately gives the stripes of the template and with a little bookkeeping, the slots can also be found.

The LCS algorithm is simple and can be made to run on long documents with acceptable memory usage, but it is not very fast. To improve the performance, we assume that the stripes are unique strings within documents. (In practice, this assumption holds for most stripes on detail pages.) With this simplification, the template is found by first finding all substrings common to all the pages but that are also unique within each page. Since the substrings might occur in a different order on each page, the algorithm then finds the longest sequence of substrings that is common to all the pages. This sequence of common substrings is the sequence of stripes of the template. Once the template is known, it is easy to find the data on a page by finding the substrings in between the stripes of the page.

Let's look at an example run of the algorithm on the following three strings:

```
<a href=yahoo.html id=0 class=bookmark>yahoo.html</a> <a
class=bookmark href=google.html id=1>google.html</a> <a
class=bookmark href=cmu.html id=2>cmu.html</a>
```

Among these three strings, the set of common unique token sequences is {`.html</a>`, `.html` `id=`, `</a>`, `<a`, `=bookmark`, `>`, `bookmark`, `class=bookmark`, `href=`, `html` `id=`, `html</a>`, `id=`}. Next for each input string, a non-overlapping set of occurrences of these sequences is found. This determines an ordering of the token sequences for each string:

```
<a, href=, .html id=, class=bookmark, >, .html</a> <a,
class=bookmark, href=, .html id=, >, .html</a> <a, class=bookmark,
href=, .html id=, >, .html</a>
```

The LCS of these three sequences is (`<a, href=, .html id=, >, .html</a>`). So the template is:

```
<a [0] href= [1] .html id= [2] >
[3] .html</a>
```

Using the template, the data can be extracted into a table:

	yahoo	0 class=bookmark	yahoo
class=bookmark	google	1	google
class=bookmark	cmu	2	cmu

3.1.2 Templates for Lists

Another common substructure of documents is tabular structure. In addition to reusing the same structure for rows, tables also group together similar items in columns.

Tables are typically used when the results of a query that retrieves more than one record are to be displayed. Multiple records are displayed in a list whose rows are generated by iterating over the records. The list is usually plugged into an outer template.

Just as a page shares a common structure with pages of the same page-type, each row shares a common structure with the other rows in the same list, and the template idea applies equally well to the rows of a list as it does to pages. Let us call the template representing the common structure of rows in a list a *row template*. We extend the basic template model so that slots are either *content slots* as before, or *list slots*, which are containers for row templates.

As an example, let's create a template to generate html segments like the following segment:

```
<h2>Cities in Pennsylvania</h2> <ul>
  <li>Aaronsburg</li>
  <li>Aaronsburg</li>
  <li>Abbottstown</li>
  <li>Abington</li>
  <li>Ackermanville</li>
</ul>
```

The outer template generates the first, second and last lines of the segment and "calls" the inner template (represented by a double box) to generate the elements of the list:

```
<h2>Cities in [ state ]</h2> <ul>
  [[ cities ]]
</ul>
```

The inner template, `cities`, simply generates one row of the list.[1]

```
<li>[ city ]</li>
```

The problem of inducing a template from a set of html pages becomes much harder with the introduction of list slots. To simplify the problem, we work on the document object model (DOM) tree of pages and make the following assumptions: The nodes representing the rows of a list are the children of a single parent node, and there can be at most one list within the children of a single parent node.[2]

Before going into the details of the template induction algorithm, we describe templates in the domain of trees. Given a set S of sequences of DOM elements, the TreeTemplate algorithm finds the LCS of S and then for each node in the LCS, builds a set of sequences of elements by gathering the child nodes of the

[1] In practice, more information needs to be specified before pages can be generated. In particular, in addition to the slots being linked to the underlying data source, a method for determining the set of cities for a particular instantiation of `state` needs to be specified.

[2] This assumption does not necessarily hold. One of the goals of the proposed approach is to be able to use this same kind of substructure while relaxing these assumptions.

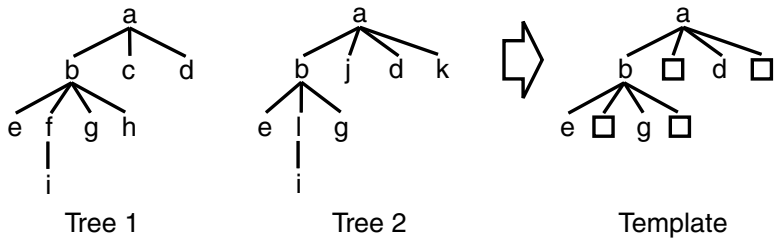

Figure 5: Templates for Trees

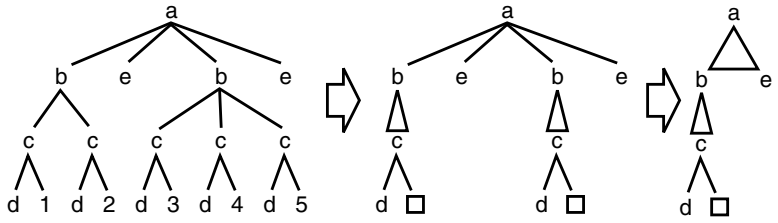

Figure 6: Row Templates

node from every tree, and calls itself with that set. For example, given the root nodes of Tree 1 and 2 in figure 5, the algorithm first finds the LCS of the one element sequences [**a**] and [**a**] and then proceeds down to the child sequences [**b c d**] and [**b j d k**]. The LCS of these two is [**b d**], so it first recurses down to the child sequences of the two **b** nodes and then to those of the two **d** nodes to get [**e g**] and []. The template is the tree of LCSs found at each node. Once the template is computed, it can be used to extract the data in the slots. This is done by finding the nodes in the original DOM structures that are not in the stripes of the template. The nodes that fall into the slots are **c**, **f**, **h**, and **i** for Tree 1, and **j**, **k**, **l**, and **i** for Tree 2. By keeping track of the slots that the nodes go into, the data can be aligned in columns and represented in a table:

f i	**h**	**c**	
l i		**j**	**k**

We now return to the template induction algorithm. HierarchicalTemplate has two steps. In the first step, it processes the DOM tree of each page to find all the lists and the corresponding row templates. In the second step, it finds the page template.

The first step processes the tree bottom-up. At each node, the algorithm looks for a repeating pattern among the child nodes. This is done by using templates as a similarity measure between two DOM subtrees. (Similar subtrees will have templates with more stripes than those that are not similar.) In particular, the algorithm looks for consecutive sequences of nodes whose similarity measure is above a threshold. Those sequences that pass the similarity test are assumed to be the rows of a list and are used as inputs to TreeTemplate to induce a

row template. As the algorithm traverses up towards the root, it replaces the repeating sequences with row templates. For example, given the tree on the left in figure 6, the algorithm first finds row templates for the inner lists within the children of **b** nodes. The resulting tree is shown in the middle of the figure, where the row template is denoted by a triangle (instead of a line) connecting it to its parent. Next, the algorithm examines the children of node **a**. With repeating subtrees represented as row templates, the algorithm can detect the similarity between the first two and the last two children of node **a** and induce the row template as shown on the right. We will refer to this intermediate tree as a row-template tree.

The second step of the induction algorithm merges the row-template trees using a variation of the basic template induction algorithm, in which row template nodes are matched only with row template nodes and regular nodes only with regular nodes.

Another way to look at templates is to consider the web-publishing system as a function that takes as input relational data and outputs a set of hyperlinked pages. Recovering the relational data from the set of hyperlinked pages is equivalent to estimating this function with a template, and inverting it to infer the underlying data.

3.1.3 Utilizing Multiple Types of Substructure

The template substructure of pages is useful in extracting data from a set of pages that are all of the same page-type. But let us look at the case when the pages are not of the same page-type. If we knew the page-type of each page, then we could easily find the template for each page-type. Conversely, if we knew the template for each page-type, then we could determine the page-type of each page. Lacking both pieces of information, clustering is a logical direction to follow, so we look at it next.

The obvious approach to clustering web pages is to use templates as a similarity measure and apply a standard clustering algorithm. The problem with this is that among pages from a single web site, there is enough commonality that the differences are usually outweighed. In fact, some clusters of pages which are easily identified by other types of substructure, such as links in a list, can be found with the template substructure only if the the pages happen to share a template among them but with not other pages.

An alternative, but ad hoc, approach is to use the template substructure in evaluating the clusters which are found by other means. The advantage of this approach is that it allows a second substructure to be used as the means to search through the cluster space. In the prototype implementation, the pages are clustered into three groups: navigation pages, detail pages, and "other" pages. The clustering algorithm generates hypotheses based on the link structure of the site (for example, navigation pages are visited before detail pages, possibly point to one or two other navigation pages, etc.) and evaluates the resulting clusters with a minimum description length (MDL) metric based on templates.

The template-based MDL metric is derived from the observation that a

template can be used to compress a set of pages. After a template is induced, it is possible to factor out the stripes from each page and encode the set as the template and a set of slots. If the template has enough content in its stripes, the resulting encoding reduces the number of bits required to represent the pages without loss of information. This idea can easily be turned into an evaluation metric for clusters where the goal is to minimize the number of bits to represent all the pages. The resulting MDL metric balances between the two extremes of having a single cluster of all the documents but with a template that has no stripes, and having one cluster for every document but with each template fully covering its document.

3.1.4 AutoWrap Implementations

AutoWrap Version 1 The first version of AutoWrap is an implementation of the HierarchicalTemplate algorithm. It takes as input a set of pages, builds a hierarchical template on the DOM trees of the pages, uses the template to extract data, and outputs the extracted data in relational form. Appendix B contains a set of example pages and the corresponding output of AutoWrap V1.

Because only one template is induced for the given set of pages, it is crucial that the input consists of pages that are of the same page-type. A single page that does not fit the template causes the system to induce an empty template and an empty template in turn causes the tokens of each page to be extracted as one large slot. This implies that the pages need to be classified very accurately, which is a difficult task even when done manually.

In the normal case where the input set consists of pages of the same page-type, AutoWrap V1 is still limited by the assumptions of HierarchicalTemplate. In particular, there is a significant number of cases where a single parent node contains more than one list among its children. This happens when a small number of lists with different row templates are put together under one parent node. AutoWrap V1 picks the list whose rows are most similar to each other and finds a row template for it, but leaves the nodes for the other lists untouched. As a result, some of the lists are never discovered. For example, in the sequence [**a 1 a 2 a 3 x b b b b**], AutoWrap would discover the list of **b**'s but not the list that contains the **a**'s.

AutoWrap V1 is not very good at finding the correct row breaks. This is because it relies only on the template-based similarity between rows. This measure is useful for identifying repeating sequences of nodes, but not for determining the boundaries of the unit of repetition. For example, just by looking at the similarty between consecutive sequences in the sequence [**a, b, c, a, b, c, a, b, c, a**], it is impossible to tell if each rows is [**a, b, c**] or [**b, c, a**].

AutoWrap Version 2 The second version of AutoWrap is a prototype implementation that utilizes the link and template substructures of a web site.

Instead of taking a set of pages, it takes a single entry URL[3] as input. Its built-in web spider creates a map of the pages connected to the entry page. After the map is created, the system starts looking for a clustering of the pages into the three clusters "navigation", "detail" and "other". The clustering that gives the best template-based compression is chosen and data is extracted from the detail pages using the induced flat template. Appendix C contains an example site, and the data extracted from the detail pages.

AutoWrap V2 suffers from the fact that it clusters pages into three predefined groups. This means that the spidering step needs to be controlled so that most of the retrieved pages belong to either the "navigation" or "detail" group. Otherwise, picking the navigation and detail pages among a large set of "noise" pages becomes difficult and both performance and quality degrade quickly.

One of the motivating factors for this work is that even though it is easy to put together systems that handle particular types of web sites by focusing on particular types of structures, these systems do not generalize well to other types of web sites. Extending them in ad hoc ways lead to inelegant systems that are still limited to a few different types of web sites. Our goal is to create elegant and flexible solutions for handling a wide variety of web sites.

3.2 Automatic Record Segmentation

As we argued above, many Web sites that present information contained in databases follow a *de facto* convention in displaying information to the users and allowing them to navigate the results. This convention gives us additional information we can leverage for structured information extraction. Figure 7 shows example list and detail pages from the Verizon Superpages site. The Superpages site allows customers to search over 16 million yellow page listings and a national white pages database by name, phone number or business type. As shown in the figure, the results returned for a search include the fields, name, address, city, state, zip and phone. Here the text "More Info" serves as a link to the detail page. Note that list and detail pages present two views of the record. Using automatic techniques, we can potentially combine the two views to get a more complete view of the record. For example, maps of the addresses are shown on the detail pages in Figure 7, but absent from the list pages.

Consider a typical list page from a Web site. As the server constructs the list page in response to a query, it generates a header containing the company logo, followed in most cases by an advertisement, then possibly a summary of the results, such as ``Displaying 1-10 of 214 records.'', table header and footer, followed by some concluding remarks, such as a copyright information or navigation aids. In the above example, the header includes `Results`, `3 Matching Listings`, `Search Again` and the associated HTML. The footer includes advertisement and navigational links. We call this part of the page the *page template*. The *page template* (Section 3.1.1) of a list page contains data

[3]In case the URL is not sufficient to retrieve the page, the contents of the entry page can also be input.

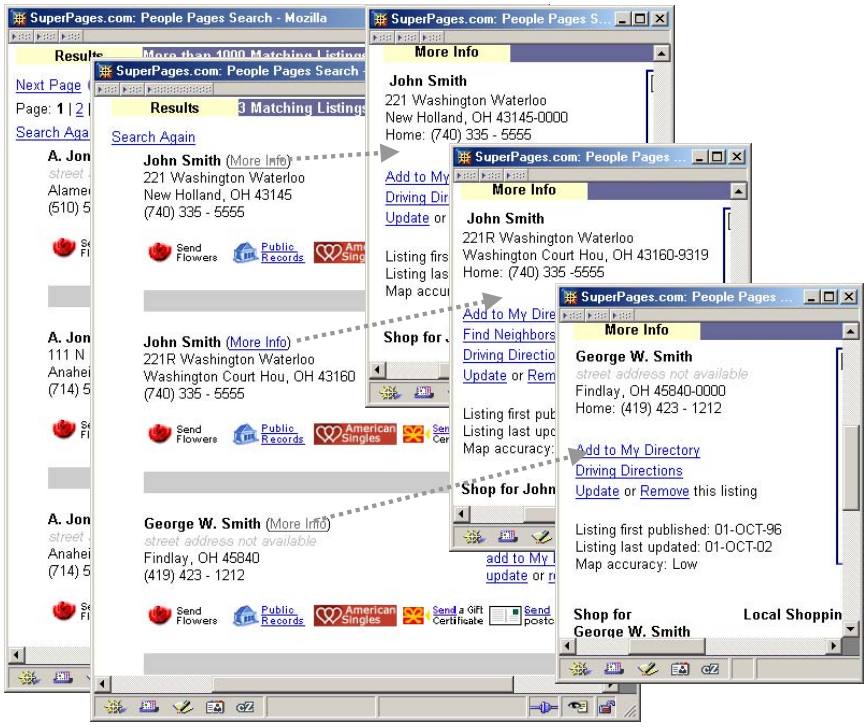

Figure 7: Example list and detail pages from the Superpages site (identifying information has been removed to preserve confidentiality).

that is shared by all list pages and is invariant from page to page. A different page template is used to generate detail pages. As the server writes out records to the table, it uses a *table template*. This template contains layout information for the table, that is the HTML tags or ASCII characters used to separate columns and rows, or format attribute values.

Given two, or preferably more, example list pages from a site, we can derive the template used to generate these pages and use it to identify the table and extract data from it. The process starts with a set of list pages from a site and a set of detail pages obtained by following links from one of the list pages. The pages are tokenized — the text is split into individual words, or more accurately tokens, and HTML escape sequences are converted to ASCII text. Each token is assigned one or more syntactic types [25, 26], based on the characters appearing in it. The three basic syntactic types we consider are: HTML, punctuation, and alphanumeric. In addition, the alphanumeric type can be either numeric or alphabetic, and the alphabetic can be capitalized, lowercased or allcaps. This gives us a total of eight (non-mutually exclusive) possible token types. A template finding algorithm (e.g., , one described in [3, 26]) is used to extract the main page template.

Slots are sections of the page that are not part of the page template. If any of the tables on the pages contain more than two rows, the tags specifying the structure of the table will not be part of the page template, because they will appear more than once on that page. Likewise, table data will also not be part of the page template, since it varies from page to page. Therefore, the entire table, data plus separators, will be contained in a single slot. Considering that tables usually contain a significant amount of data, we use a heuristic that the table will be found in the slot that contains the largest number of text tokens.

Next, we extract data from the table. We do this simply by extracting, from the slot we believe to contain the table, the contiguous sequences of tokens that do not contain separators. Separators are HTML tags and special punctuation characters (any character that is not in the set ".,()-"). Practically speaking, we end up with all visible strings in the table. We call these sequences *extracts*, $E = \{E_1, E_2, \ldots, E_N\}$. These are the attribute values that the records in the table contain, and possibly some extraneous data, such as "More Info", "Send Flowers" as shown in Figure 7. Our goal is to segment these extracts into individual records.

The CSP and probabilistic approaches share the same basic premise: detail and list pages present two views of the same record and each detail page corresponds to a distinct record. The labels for detail page are: $\{r_1, r_2, \ldots, r_K\}$.

For each extract E_i, we record all detail pages on which it was observed, D_i. [4] The extracts are checked in the order they appear in the text stream of the list page. If an extract appears in all the list pages or in all the detail pages, it is ignored: such extracts will not contribute useful information to the record segmentation task.

[4]The string matching algorithm ignores intervening separators on detail pages. For example, a string "FirstName LastName" on list page will be matched to "FirstName
 LastName" on the detail page.

The methods presented below are appropriate for tables that are laid out horizontally, meaning that the records are on separate rows. A table can also be laid out vertically, with records appearing in different columns; fortunately, few Web sites lay out their data in this way. In horizontally laid out tables, the order in which records appear in the text stream of the page is the same as the order in which they appear in the table. In other words, any attribute of the second record will appear in the stream after all the attributes of the first record and before any of the attributes of the third record have been observed.

Table 2 is an example of a table of observations of extracts on detail pages from the Superpages site (see Figure 7).[5] The columns correspond to extracts and they are displayed in the order they appear on the list page.

As can be seen from the table, the same extract can appear on multiple detail pages. In the Superpages site, for example, several people may have the same name or the phone number. Note that only the extracts or record attributes that appear in both the list and detail pages are used. There may be other potential attributes, but if they do not appear on any of the details pages, they will not be considered in the analysis. This works to our advantage by reducing the need to precisely identify the table slot.

	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}	E_{11}
	John Smith	221 Wa ington...	New Holland...	(740) 335-5555	John Smith	221R Wa shington...	Wash ington...	(740) 335-5555	George W. Smith	Findlay, OH...	(419) 423-1212
D_i	r1,r2	r1	r1	r1,r2	r1,r2	r2	r2	r1,r2	r3	r3	r3

Table 2: Observations of extracts on detail pages D_i for the Superpages site

Extracts common to list and detail pages give us an additional source of information that we can use to segment data into individual records. An extract (attribute value) belongs to a record only if it appears on the detail page corresponding to that record. The same extract cannot be assigned to more than one record or more than once to the same record. Thus, in the table above, we can use these rules to assign E_1, E_2, E_3 and E_4 to the first record, and E_5, E_6, E_7 and E_8 to the second record, as shown in Table 3, even though E_1 and E_4 appear in both records. These rules can be easily encoded in both the CSP and the probabilistic framework. Solving this problem yields an assignment of data to records.

The probabilistic model is more expressive than the CSP. In addition to record segmentation, we can learn a model for predicting the column of an extract, based not only on its token type, but also on the neighboring columns. We can learn a probabilistic model of the data given the column label: i.e., , first attribute of the record is of alphabetic type, the second a numeric type, etc. The inference algorithm estimates parameters of the model from the observations of extracts on details pages. The parameters are then used to find a column assignment that maximizes the total probability of the observations given the model. The column labels will be $L_1, \ldots, L_k$ (we can find k by the longest

[5] All identifying information has been changed to preserve anonymity.

potential sequence of r_i in the data). To provide them with more semantically meaningful labels, we can use other automatic extraction techniques, such as those described in the Roadrunner system [4].

	E_1 John Smith	E_2 221 Wa ington...	E_3 New Holland...	E_4 (740) 335-5555	E_5 John Smith	E_6 221R Wa shington...	E_7 Wash ington...	E_8 (740) 335-5555	E_9 George W. Smith	E_{10} Findlay, OH...	E_{11} (419) 423-1212
r1	1	1	1	1							
r2					1	1	1	1			
r3									1	1	1

Table 3: Assignment of extracts to records

3.2.1 A CSP Approach to Record Segmentation

CSPs are stated as logical expressions, or constraints, over a set of variables, each of which can take a value from a finite domain (Boolean, integer, etc.). The case where the variables and logical formulas are boolean is known as Boolean satisfiability, the most widely studied area of CSP. In a pseudo-boolean representation, variables are 0-1, and the constraints can be inequalities. The CSP problem consists of finding the value assignment for each variable such that all constraints are satisfied at the same time. When constraints are inequalities, the resulting problem is an optimization problem.

Structure Constraints We encode the record segmentation problem into pseudo-boolean representation and solve it using integer variable constraint optimization techniques. Let x_{ij} be the assignment variable, such that $x_{ij} = 1$ when extract E_i is assigned to record r_j. and $x_{ij} = 0$ when the extract E_i is not part of the record r_j. The assignment of extracts to records will look something like the table in Table 3. Blank cells in table correspond to $x_{ij} = 0$. By examining this table, we see that there are two logical constraints it makes sense to impose on the assignment variables. First, there has to be exactly a single "1" in each column of the assignment table. This is the *uniqueness constraint*.

Uniqueness constraint: Every extract E_i belongs to exactly one record r_j.

Mathematically, the uniqueness constraint can be stated as
$\sum_j x_{ij} = 1$. If necessary, we can make the constraint less rigid by requesting that every extract appear in at most one record. The relaxed constraint can be written as $\sum_j x_{ij} <= 1$.

The assignment table, Table 3, suggests a second constraint, what we call the *consecutiveness constraint*.

Consecutiveness constraint: only contiguous blocks of extracts can be assigned to the same record.

These constraints can be expressed mathematically in the following way: $x_{ij} + x_{kj} \leq 1$ when there is n, $k < n < i$, such that $x_{nj} = 0$. In other words, we cannot assign extract E_1 in Table 2 to r_2 along with extracts E_4, E_5, and E_6 because neither E_2 nor E_3 can be assigned to r_2. A better choice is to assign E_1 E_2 E_3 E_4 to r_1 and E_5 E_6 E_7 E_8 to r_2.

The observations and assignment tables are closely linked. If extract E_i was not observed on detail page r_j ($r_j \notin D_i$), then $x_{ij} = 0$, in other words, E_i cannot be assigned to r_j. If E_i was observed on detail page r_j, then x_{ij} is either 1 or 0. The constraints on the assignment variables can be easily written down from the observations data. For the Superpages site data shown in Table 2 the uniqueness and consecutiveness constraints are written below:

$$\begin{array}{ll} x_{11} + x_{12} = 1 & x_{11} + x_{81} <= 1 \\ x_{21} = 1 & x_{21} + x_{81} <= 1 \\ x_{31} = 1 & x_{31} + x_{81} <= 1 \\ x_{41} + x_{42} = 1 & x_{31} + x_{81} <= 1 \\ x_{51} + x_{52} = 1 & x_{41} + x_{81} <= 1 \\ x_{62} = 1 & x_{12} + x_{42} <= 1 \\ \vdots & \vdots \end{array}$$

	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8
pos_1^{730}	1				1			
pos_1^{772}		1						
pos_1^{812}			1					
pos_1^{846}				1				1
pos_2^{536}	1				1			
pos_2^{578}				1				1
pos_2^{608}						1		
pos_2^{642}							1	

Table 4: Positions of extracts on detail pages. Entry of 1 means extract E_i was observed at position k on page j (pos_j^k).

Position Constraints Detail pages present another source of constraints we can exploit, because in addition to the occurrence of an extract on a page, they provide information about the position of that extract. This information is summarized in Table 4 for the Superpages site. The horizontal rows are the positions on page j (e.g., , token number of the starting token) where the extracts were observed. Note that in Table 4 the first four positions are from detail page r_1, while the next four are from r_2. If extract E_i was observed in position $pos_k^j(E_i)$ on detail page r_k, the (i, j) cell in table has an entry "1"; otherwise, it is empty. It is clear that no two extracts assigned to the same

record can appear in the same position on that page. The corollary is: if two extracts appear in the same position on the detail page, they must be assigned to different records. We express it more formally as

Position Constraint: if $pos_j(E_i) \neq pos_j(E_k)$, then E_i and E_k cannot both be assigned to r_j

In the example in Table 4, the position constraints are

$$x_{11} + x_{51} = 1$$
$$x_{12} + x_{52} = 1$$
$$x_{41} + x_{81} = 1$$
$$\vdots$$

These constraints can also be relaxed to produce inequalities.

After we have constructed the uniqueness, consecutiveness and position constraints on the assignment variables for a particular Web page, we solve them using WSAT(OIP) [41], an integer optimization algorithm [42]. The solution is the assignment of extracts to records. Results are presented in Section 3.2.4.

3.2.2 A Probabilistic Approach to Record Segmentation

An alternate approach is to frame the record segmentation and extraction task as a probabilistic inference problem. Common probabilistic models for information extraction include hidden Markov models (HMMs) [35], and conditional random fields (CRFs) [22, 33]. In these approaches, a labeled training set is provided and the model is learned using standard probabilistic inference techniques; because the state is *hidden*, the common approach to learning the models is to use the expectation maximization (EM) algorithm. While these approaches have their computational limitations, they have been applied successfully to a wide range of problems beyond information extraction including speech recognition and robot navigation.

Unfortunately, here we are faced with a more challenging problem. We do **not** have a labeled training set to start from. The key to our success will be to:

Factor: We will factor the state space and observation set of the HMM to allow for more efficient learning (because fewer parameters will be required).

Bootstrap: We will use the information from the detail pages to help bootstrap the learning algorithm. The constraints from the detail extracts will provide useful information that can keep our learning algorithm on track.

Structure: We will use a hierarchical model to capture global parameters such as the length of the record, or the period, to make our inference more tractable. Note that while there is a global record length, the record lengths of the individual records may vary; for some records not all columns will be presented.

We begin by describing the probablistic model that we will use (Section 3.2.2), then describe how the model is used to do record segmentation. (Section 3.2.3).

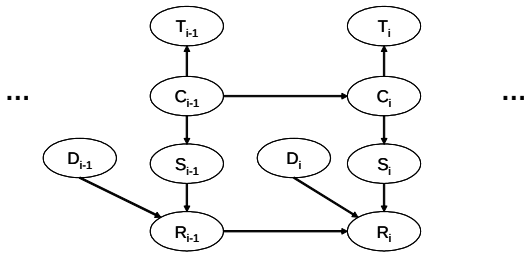

Figure 8: A probabilistic model for record extraction from list and detail pages.

Probabilistic Model for Record Extraction Figure 8 shows a graphical model representation for our problem. The basic variables in the model are the following:

$T = \{T_1, \ldots, T_n\}$: token types of extract E_i. Examples of token types are alphanumeric, capitalized, punctuation, as described earlier. We have 8 token types, so each T_i is represented as a vector $T_{i_1}, T_{i_2}, \ldots, T_{i_8}$.

$D = \{D_1, \ldots, D_n\}$: record numbers of the detail pages $D_i \subseteq 1, \ldots, K$ on which E_i occurred.

The above variables are *observed* in our data — given a list and set of detail pages, we can compute the values for the above variables.

In addition, we have the following set of *unobserved* variables:

$R = \{R_1, \ldots, R_n\}$: the record number of the extract. We assume these range from $1 \ldots K$ (these correspond to detail pages).

$C = \{C_1, \ldots, C_n\}$: the column label of the extract. We assume these range from $L_1, \ldots, L_k$ (a bound on this is the largest number of extracts found on a detail page).

$S = \{S_1, \ldots, S_n\}$: S_i is true if E_i is the start of a new record, false otherwise.

Of course, in addition to the variables, our model describes the dependencies between them. Rather than using the standard HMM representation, we use a factored representation [16, 29], which allows us to more economically model (and learn) the state transition probabilities. We have defined the factored structure of the model, as shown in Figure 8. Arrows indicate probabilistic dependencies. In some cases these dependencies are deterministic — for example if an extract appears on only one detail page, then we know the record to which it belongs. In other cases the dependencies are probabilistic — for example, the starting token type of a column might usually be a *capitalized* string, but occasionally will be an *HTML* tag. Here, we assume the structure of the dependencies (and where reasonable, the functional form for the dependencies) and we will learn the probabilities for the model using our observed data.

Our model assumes the following dependencies:

$P(T_i|C_i)$: The token type for extract i depends on the column label. For example, for the name field, the token type is likely to be *capitalized* token, but this is a probabilistic relationship.

$P(C_i|C_{i-1})$: The column label for extract i depends on the column label of the previous column. For example, the *address* field is likely to follow the *name* field. Note that because the token type in turn depends on the column label, it will also provide information to help us determine the column label. For example, if the previous column label is *name*, and the current token is *numeric*, we may think the column label *address* is most likely. However if the token is *all-caps*, we might think the column label is more likely to be *state*. Note that while we used the values *name* and *address* above, we really only have the column labels $L_1, \ldots, L_k$. As mentioned earlier, we may be able to automatically create semantically meaningful names for them using other automatic extraction techniques.

$P(S_i|C_i)$: S_i, whether extract i starts a new record, depends on the column label. It turns out that while later columns may be missing from a record, in all of the domains that we have examined the first column, which usually contains the most salient identifier, such as the Name, is never missing. This allows us to make a very useful simplification: rather than needing to learn the transition probabilities for the data, it makes sense to assume a deterministic relationship: $P(S_i = \text{true}|C_i = L_1) = 1.0$ and $P(S_i = \text{true}|C_i = L_j, j \neq 1) = 0$. Note that since C_i is not observed, in order to do segmentation we will still need to do probabilistic inference to compute C_i.

$P(R_i|R_{i-1}, D_i, S_i)$: The record number for extract i will depend on the record number of the previous extract, whether or not S_i is the start of a new record, and D_i, the detail pages on which E_i has been observed. In general, this is a deterministic relationship: if S_i is false, then $P(R_i = R_{i-1}) = 1.0$ and if S_i is true, then $P(R_i = R_{i-1} + 1) = 1.0$ However D_i also constrains R_i. R_i must take one of the values of D_i.

The task of record segmentation boils down to finding values for the unobserved R and C variables. As is commonly done in probabilistic models for sequence data, we compute maximum a posteriori (MAP) probability for R and C and use this as our segmentation:

$$\arg\max P(R, C|T, D)$$

Because we are assuming a Markov model, we can write:

$$P(R, C|T, D) \quad = \quad \prod_{i=1}^{n} P(R_i, C_i|D_i, T_i, R_{i-1}, C_{i-1})$$

and using the structure of the graphical model above, this simplifies to:

$$P(R, C|T, D) = \prod_{i=1}^{n} \sum_{S_i} P(R_i|D_i, S_i)P(S_i|C_i)P(C_i|T_i)$$

3.2.3 Learning the Model

At this point, we could apply a standard off-the-shelf probabilistic inference algorithm to learn the model parameters and to make the appropriate inferences. Unfortunately, our model is so unconstrained, we would have little luck inferring anything useful. We will use two ideas: **bootstrapping** and **structure** to make successful inferences.

Bootstrapping The key way in which information from detail pages helps us is it gives us a guide to some of the initial R_i assignments. It turns out this little bit of information can be surprisingly informative.

Recall that D_i is the set of detail pages on which extract E_i occurs. This provides invaluable evidence for the record assignment for R_i. We make use of this information by setting:

$$P(R_i = r_i) = \frac{1}{|D_i|}$$

and $P(R_i = r_i) = 0)$ for all $r_i \notin D_i$.

In addition, we make the following initial assumptions for the parameters of the model: $P(T_{i_j} = \text{true}|C_i) = 1/8$, in other words, without observing any of the data, we assume that the probability of a token type occurring, given the column, is 1/8. Note that this does not preclude an extract being of more than one token type. While we begin with this uniform assumption, we will be updating the model parameters as we go along, so the parameters will quickly be updated in accordance with the frequency that we observe the different tokens and column labels.

We also make use of the D_i to infer values for S_i. If $D_{i-1} \cap D_i = \emptyset$, then $P(S_i = \text{true}) = 1$. As a simple example, if extract i only appears on detail page j and extract $i - 1$ only appears on detail page $j - 1$, then $S_i = \text{true}$.

Structure Besides the information from the detail pages, another important piece of information we can make use of is the record length, or the period π of the table; π is the number of columns in the table. Recall, however, that not every record will have all π columns. Instead, for each record r_j, there will be an associated π_j, which is the number of fields in record j. This approach allows for missing fields in a record — a common occurrence in Web data.

One way of allowing this is simply to make use of the S_i. If $S_i = \text{true}$ and $R_i = j$ and $S_{i'}$ is the next record start indicator that is true, then we can simply compute $\pi_j = i - i'$. But this fails to capture the correlations among record lengths. In our example, there are 4 fields possible in a full record, perhaps

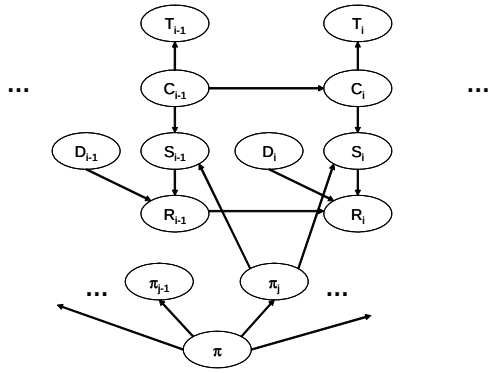

Figure 9: A probabilistic model for record extraction from list and detail pages which includes a record period model π.

most often we will have all 4 fields, Name, Address, City, Phone, but another common occurrence is to just have the 3 fields Name, City and Phone. We want to be able to learn this. Now, the columns C_i will be conditioned on the corresponding π_j. We can learn for example that $C_i = City$ is much more likely if $\pi_j = 3$ (and $C_{i-1} = Name$).

The difficulty here is that until we know the correspondence of extract i to r_j; we do not even know which π_j to depend on. This could make inference even more complex. It turns out however, that there are only a small number of possibilities, so this is in fact feasible. Furthermore this more complex model does in fact give us improvements in accuracy. Figure 9 shows a graphical model representation of the updated model.

Implementation We use EM to implement the probabilistic inference algorithm. We have developed a variant of the forward-backward algorithm that exploits the hierarchical nature of the record segmentation problem. By explicitly modeling the period probabilities, we can constrain the structure of the model and in turn use this structure to guide the inference process.

The basic components of the algorithm are:

1. Compute initial distribution for the global period π using the current values for the S_i. Because of the constraints the detail pages offer, our initial S_i will provide us with some useful information for π.
2. Now, having updated π, we compute the π_k for each record j.
3. For each potential starting point and record length, we update the column start probabilities, $P(C_i|T_i, C_{i-1})$.
4. Next we update $P(S_i|C_i)$.
5. And finally we update $P(R_i|R_{i-1}, D_i, S_i)$

In the end we output the most likely assignment to R and C. This gives us the segmentation for the list page. While in theory, the running time of this algorithm is potentially exponential in the length of the extract sequence, by using the period probabilities to structure the inference we get enormous savings. In practice the algorithm is very efficient and took just a few seconds to run on each of our test cases.

3.2.4 Results

We now present results of automatic segmentation of records using the CSP and probabilistic approaches described in the sections above.

Experimental Setup The data set consisted of list and detail pages from 12 Web sites in four different information domains, including book sellers (*Amazon*, *BNBooks*), property tax sites (*Buttler*, *Allegheny*, *Lee* counties), white pages (*Superpages*, *Yahoo*, *Canada*411, *SprintCanada*) and corrections (*Ohio*, *Minnesotta*, *Michigan*) domains. From each site, we randomly selected two list pages and manually downloaded the detail pages. In this work, we were not concerned with the task of automatically determining detail pages. In fact, there are often other links from the list page that point to advertisements and other extraneous data. Such data may or may not influence the segmentation results. In future work, we will attempt to automate detail page identification. One approach is to use heuristic rules, such as "follow links in the table" or "follow a link whose text field is More Info." Alternatively, one can download all the pages that are linked on the list pages, and then use a classification algorithm to find a subset that contains the detail pages only. The detail pages, generated from the same template, will look similar to one another and different from advertisement pages, which probably don't share any common structure.

In addition to displaying different data, the pages varied greatly in their presentation and layout. Some used grid-like tables, with or without borders, with easily identifiable columns and rows. Others were more free-form, with a block of the page containing information about an item, followed by another block containing information about another item. Within the block, attributes could appear in separate columns, rows, or in a formatted table. The entries could be numbered or unnumbered. Commercial sites had the greatest complexity and more likely to be free-form than government sites.

The CSP and probabilistic algorithms were exceedingly fast, taking only a few seconds to run in all cases.

3.2.5 Evaluation

Table 5 shows results of automatic record segmentation for these 12 sites using two different approaches, probabilistic and constraint satisfaction. Both approaches share a common step, the page template finding algorithm. In cases where the template finding algorithm could not find a good page template, we

	Probabilistic				CSP				
Wrapper	**Cor**	**InC**	**FN**	**FP**	**Cor**	**InC**	**FN**	**FP**	**notes**
Amazon	4	6	0	1	0	0	10	0	a, b
Books	2	5	3	4	0	0	10	0	
BN	5	5	0	0	2	0	8	0	a, b, c, d
Books	5	5	0	0	0	0	10	0	
Allegheny	20	0	0	0	20	0	0	0	
County	16	4	0	0	20	0	0	0	
Butler	15	0	0	0	15	0	0	0	
County	12	0	0	0	12	0	0	0	
Lee	16	0	0	0	16	0	0	0	
County	5	0	0	0	5	0	0	0	
Michigan	7	0	0	0	4	3	0	0	
Corrections	12	4	0	0	2	8	6	0	c, d
Minnesota	11	0	0	0	4	7	0	0	a, b, c, d
Corrections	17	2	0	0	8	9	0	2	
Ohio	8	2	0	0	10	0	0	0	
Corrections	10	0	0	0	10	0	0	0	
Canada	18	7	0	0	25	0	0	0	
411	1	4	0	0	1	4	0	0	c, d
Sprint	17	3	0	0	20	0	0	0	
Canada	8	12	0	0	20	0	0	0	
Yahoo	0	10	0	0	5	5	0	0	a, b, c, d
People	10	0	0	0	10	0	0	0	b
Super	3	0	0	0	3	0	0	0	a, b
Pages	9	6	0	0	15	0	0	0	
Precision	0.74				0.85				
Recall	0.99				0.84				
F	0.85				0.84				

Notes
a. Page template problem; b. Entire page used; c. No solution found;
d. Relax constraints

Table 5: Results of automatic record segmentation of tables in Web pages using the probabilistic and CSP approaches

have taken the entire text of the list page for analysis. Only the strings that appeared on both list and detail pages were used in record segmentation. The rest of the table data are assumed to belong to the same record as the last assigned extract. The reason for this is that each row of the table usually starts with the most important attribute, such as the name or ID number. This attribute will almost always appear on the detail page as well.

We manually checked the results of automatic segmentation and classified them as correctly segmented (Cor) and incorrectly segmented (InCor) records, unsegmented records (FN) and non-records (FP). Precision and recall are defined below. We used the F measure to gauge the accuracy of the task.

$$\begin{aligned} P &= Cor/(Cor + InCor + FP) \\ R &= Cor/(Cor + FN) \\ F &= 2PR/(P + R) \end{aligned}$$

We calculated $P = 0.74$, $R = 0.99$ and $F = 0.85$ for the probabilistic approach and $P = 0.85$, $R = 0.84$ and $F = 0.84$ for the CSP approach. This is an exceedingly good performance for automatic algorithms. Using heuristics, as described below, we can further improve on the results.

Each approach has its benefits and drawbacks, which make them suitable in different situations. The CSP approach is very reliable on clean data, but it is sensitive to errors and inconsistencies in the data source. One such source of data inconsistency was observed on the Michigan corrections site, where an attribute had one value on the list pages and another value on the detail pages. This by itself is not a problem; however, the list page string appeared on one detail page in an unrelated context. The CSP algorithm could not find an assignment of the variables that satisfied all the constraints. The probabilistic approach, on the other hand, tolerates such inconsistencies and is more expressive than the CSP representation. Its expressiveness gives us the power to potentially assign extracts to individual attributes, and, when combined with a system that automatically extracts column labels [4] from tables, reconstruct the relational database behind the Web site. Both techniques (or a combination of the two) are likely to be required for robust and reliable large-scale information extraction. We stress that the approaches are novel in that they are domain independent, unsupervised, and rely on the content of Web pages rather than their layout.

The page template finding algorithm performed poorly on five of the 12 sites: *Amazon, BnBooks*, *Minnesota Corrections*, *Yahoo People* and *Superpages*. In the first three sites, the entries were numbered. Thus, sequences such as "1.", will be found on every page. If the tables are of different lengths, the shortest table will limit what is to be considered a page template, and the remainder of data on the longer tables will be extracted. When we encountered a problem with the page template algorithm, we use the entire page as the table slot — in other words, we used the entire content of the list page for matching with the detail page. At times, using the entire list page led to problems (*Amazon*, first *Yahoo People* list page), as many strings that were not part of the table found matches on detail pages, although in some cases this approach performed quite

well (second list in *Yahoo People*, *Superpages*). There are a number of ways to circumvent this problem which were not explored in this paper. One method is to simply follow the "Next" link, and download the next page of results. The entry numbers of the next page will be different from others in the sample. Another approach is to build a heuristic into the page template algorithm that finds enumerated entries. We will try this approach in our future work.

The CSP approach performed extremely well on clean data; however, it was prone to fail when there were errors and inconsistencies in the data underlying the Web sites. For example, on one *Canada411* page, one of the records had the town attribute missing on the detail page but not on the list page. Since the town name was the same as in other records, it was found on every detail page but the one corresponding to the record in question. As a result, WSAT(OIP) could not find a solution satisfying all constraints. Relaxing constraints by replacing equalities with inequalities produced a solution, but it was a partial solution, because not every extract was assigned to a record. Another common data inconsistency that caused WSAT(OIP) to fail was when the attribute had different values on list and detail pages. For example, on the *Amazon* site, a long list of authors was abbreviated as "FirstName LastName, et al" on list pages, while the names appeared in full on the detail page. On the *Minnesota Corrections* site, there was a case mismatch between attribute values on list and detail pages. On the *Michigan Corrections* site, status of an paroled inmate was listed as "Parole" on list pages and "Parolee" on detail pages. Unfortunately, the string "Parole" appeared on another page in a completely different context. As a result, all constraints could not be satisfied. In such cases we relaxed the constraints, for example, by requiring that an extract appear on at most one detail page. WSAT(OIP) was able to find solutions for the relaxed constraint problem, but the solution corresponded to a partial assignment. If we excluded from consideration those Web pages for which the CSP algorithm could not find a solution, performance metrics on the remaining 17 pages were $P = 0.99$, $R = 0.92$ and $F = 0.95$. This performance is comparable to that obtained with hand-crafted heuristic or domain-specific rules. [6, 9] The probabilistic approach was less sensitive to data inconsistencies, but was slightly less accurate overall. On the same 17 pages as above, its performance was $P = 0.78$, $R = 1.0$ and $F = 0.88$.

Except for the two book sites, we were able to correctly segment at least one table using either method. The tables on the book site pages presented challenges. The entries in these lists were numbered. As a result, the page template algorithm did not work, and we had to use the text of the entire list page. Unfortunately, many of the strings in the list page, that were not part of the list, appeared in detail pages, confounding our algorithms. For the *Amazon* site the problem was further compounded by the fact that we downloaded the pages manually. The site offers the user a useful feature of displaying her browsing history on the pages. This led to title of books from previously downloaded detail pages to appear on unrelated pages, completely derailing the CSP algorithm. Even relaxing constraints to find a partial assignment did not produce good results for these two sites. These observations do not apply to the perfor-

mance of the algorithm, only the data collection and preparation steps. Adding domain-specific data collection techniques should improve the final segmentation results.

The probabilistic approach allows us to assign extracts to attributes, not only records. This remains an open problem for future research. It may also be possible to obtain the attribute assignment in the CSP approach, by using the observation that different values of the same attribute should be similar in content, e.g., , start with the same token type. We may be able to express this observation as a set of constraints.

As discussed earlier in this paper, the RoadRunner system uses an elegant approach to automatically extract data from data-rich Web sites. RoadRunner assumes that the results pages from a site were generated by a union-free grammar and induces this grammar from example list pages. This approach, however, fails for Web sites that use alternate formatting instructions for the same field. Such alternate instructions are syntactically equivalent to disjunctions, which are disallowed by union-free grammars. Consider Superpages list page in Figure 7. If an address field is missing, the text "street address not available" is displayed in gray font; otherwise, an address is displayed in black. In each alternative, different HTML tags are used to format the address field. There is still an underlying grammar that gave rise to this Superpages page, but this grammar admits unions. Inferring such grammars from examples is an even more computationally complex task than inferring union-free grammars. Our approach, on the other hand, is more computationally efficient, because rather than using a page's layout, it relies on the content of the page, which is often much less than the layout. Both of our methods handle the Superpages site very effectively, as shown in the results.

4 Automatic Labeling

The difficulties inherent in automatic record extraction are even more pronounced in the problem of semantically labeling the extracted fields. Despite the Web wrappers' long track record, automatic labeling of extracted data has only recently begun to be addressed [4]. We have developed a domain independent approach to automatically extracting records from Web sites and semantically labeling the fields, or mapping them to a schema. The work is build on our previous research in automatic generation and maintenance of Web wrappers [26]. Our starting assumption is that content on different Web sites in the same domain contains many similarities: e.g., used car sites generally give the year, make and model of the car, as well as its mileage and price. Additional information, such as the color and engine specifications may also be given. The data fields will have a similar format on different sites within the domain. If we learn the representation of fields on one site, we can apply it to label content on other sites within the domain.

The architecture of the Automatic Data Labeling (ADeL) system is shown in Figure 2. We will use the Used Cars shopping site as the running example. The

training stage consists of background knowledge acquisition, where we collect data in a particular domain, e.g., Used Cars, and learn a structural description of it. In order to collect a large number of examples, we created wrappers for some sites in the domain. We specified the schema of the data and labeled example records on several pages. We used the wrapper building tool developed by Fetch Technologies. It provides a GUI to simplify data entry and uses the wrapper induction system [31] to learn the correct wrapper extraction rules.

Next, we learn a description of the data fields. We represent the structure of a data field by sequences of tokens and token types, which we call *patterns*. We use the DataPro algorithm [26] to learn patterns from examples collected by the wrappers, as described in Section 4.1.

The next step is to extract data from pages. The pages have to be classified into separate types, such as list or detail pages, before data can then be extracted from pages of a single type. AutoWrap, the automatic data extraction algorithm is described in Section 3.

Finally, we use the patterns learned on the training examples to assign semantic labels to the automatically extracted records, as described in Section Labeling. We validated the ADeL system on the Used Cars domain. Results of this work are presented in the results section.

4.1 Modeling Data Content

The data modeling step is used to learn the structure of data fields from examples. We represent the structure of data by patterns of tokens and token types. In previous work, we developed a flexible pattern language and presented an efficient algorithm, DataPro, for learning patterns from examples of a field [25, 26]. The pattern language contains specific tokens and general token types. Specific types refer to unique text strings, such as "California" or "sea", while the general types describe the syntactic category to which the token's characters belong, such as numeric, alphabetic, etc.The token types are organized in a hierarchy, which allows for multi-level generalization.[6] The pattern language can be extended to include other syntactic types or domain-specific semantic types. In addition to patterns, we also remember the mean length of a field and its variance.

DataPro algorithm finds patterns that describe many of the examples of a field and are highly unlikely to describe a random token sequence. As an example, names can be represented as a set of patterns such as "capitalized word followed by an initial" and "capitalized word followed by a capitalized word," whereas addresses can be represented as "number followed by two capitalized words followed by the word Blvd," etc.

The symbolic representation of content by patterns of tokens and token types is very flexible and general. In our previous research we found that a set

[6] A text token is a punctuation mark (PUNCT) or an alphanumeric token (ALNUM). If it is alphanumeric, it could be alphabetic type (ALPHA) or a number (NUMBER). If alphabetic, it could also be a capitalized word (CAPS) or an all-capitalized word (ALLCAPS). The number category is further subdivided into 1DIGIT to 5DIGIT numbers.

of patterns describing how a field begins and ends, allowed us monitor wrapper's accuracy or locate examples of the field on new pages with considerable accuracy [26]. We apply patterns to recognize known data fields, as described below.

4.2 Labeling

When provided with several list pages, AutoWrap extracts all tables from these pages. These include one with the data we are interested in extracting, as well as tables containing extraneous information. The next step in the process is to label the columns of every table and output the correct table of data, which we define to be one that has the most labeled columns.

We use learned patterns to map columns to data fields. The basic premise is to check how well a field describes a column of data, given a list of patterns that describe the data field and its mean length. We have developed a set of heuristics to score how well a field describes a column. The column is assigned the field with the highest score. Factors that increase a field's score include

- Number of patterns that match examples in the column
- How close examples are in length to the field's mean length
- Pattern weight — where the more specific patterns are given a higher weight

4.3 Results

We validated the ADEL system on the Used Cars domain. We wrapped two used cars sites — Anaheim Lincoln Mercury and Mercedes Benz of Laguna Niguel — and collected on the order of 250 records from these sites. We normalized all data by lowercasing it. We then ran the DataPro algorithm on the records to learn descriptions of the fields. The resulting patterns and field lengths are displayed in Table 6. Fields Mileage and Price had many specific patterns that we do not display in the table.

Next, we attempted to extract and label data from three new sites in the Used Cars domain.

We manually collected three list pages from each new site. AutoWrap automatically induced the template for each set of three pages and extracted all data from them. AutoWrap found, on average, six tables of data in each set of pages. These tables were processed further. Again, we normalized data by lowercasing it. In addition, we fixed some of the segmentation errors AutoWrap made. For instance, AutoWrap does not recognize the sequence "`\\r\\n`" or a comma as a field delimiter. Thus, "Marina del Rey, CA" is extracted as a single field, rather than two. These problems will eventually be fixed in the AutoWrap source. For now, we manually segmented fields on "`\\r\\n`" and "," (except for numbers, in which case we did not segment them on the comma).

Year
< 1.0 ± 0.0 >
[1999]
[2002]
[2000]
[2003]
[2001]
[4DIGIT]

Make
< 1.31 ± 0.46 >
[mercedes, benz]
[ford]
[mercury]
[lincoln]

Model
< 1.88 ± 0.76 >
[s430]
[ranger 2wd]
[mountaineer ALNUM]
[gr marquis ls]
[ls v6]
[ls v8]
[navigator 2wd]
[navigator ALNUM]
[ALPHA lx]
[ALPHA ALPHA]

Color
< 1.12 + −0.32 >
[smoke silver]
[desert silver]
[silver]
[black]
[grey]
[ALPHA]

Mileage
< 2.38 ± 0.92 >
[47K]
[21 , 3DIGIT]
[20 , 3DIGIT]
[2DIGIT , 3DIGIT]

Price
< 4.0 ± 0.0 >
[$ 2 , 988]
[$ 25 , 3DIGIT]
[$ 15 , 988]
. . .
[$ 29 , 3DIGIT]
[$ 30 , 988]
[$ 31 , 3DIGIT]
[$ 2DIGIT , 995]
[$ 2DIGIT , 990]
[$ 2DIGIT , 900]

Bodystyle
< 3.06 ± 0.24 >
[4 dr sedan]
[4 dr ALPHA ALPHA]
[2 dr coupe]

Engine
< 3.83 ± 0.38 >
[5 . 0l v8]
[4 . 3l v8]
[3 . 2l 6cyl]
[3 . 2l ALNUM]
[2 . 3l]
[2 . ALNUM]

VIN
< 1.0 ± 0.0 >
[ALNUM]

Table 6: Patterns learned for data fields in the Used Cars domain

The columns were scored against the patterns in Table 6 according to the criteria described in Section Labeling. The column was assigned the label of the highest scoring field. The table with the most labeled columns was output in comma-delimited spreadsheet format. Table 5 shows a section of a table of labeled data for one site.

-	Model	Color	Color	Bodystyle	-
1998 acura	2 . 5tl	automatic	silver	4 dr sedan	2.5 liter 5 cyl.
2001 acura	3 . 2cl	automatic	red	2 dr coupe	3.2 liter 6 cyl.
2000 acura	3 . 2tl	automatic	white	4 dr sedan	3.2 liter v-6
2000 acura	integra gs	5 speed manual	silver	2 dr hatchback	1.8 liter 4 cyl.
1996 acura	integra ls	5 speed manual	purple	4 dr sedan	1.8 liter 4 cyl. dohc
1998 chevrolet	astro	automatic	orange	mini van	4.3 liter v-6
2001 chevrolet	camaro	automatic	blue	2 dr coupe	3.8 liter 6 cyl.

VIN	Mileage	Mileage	Model
jh4ua265xwc007394	60 , 913	14 , 976	norm reeves
19uya42741a003823	76 , 674	19 , 934	norm reeves
19uua5668ya054953	41 , 421	19 , 595	norm reeves
jh4dc4368ys014324	36 , 866	16 , 895	norm reeves
jh4db7553ts011459	62 , 723	9 , 595	norm reeves
1gndm19w0wb180216	83 , 604	8 , 903	cerritos ford
2g1fp22k412109568	22 , 249	13 , 975	norm reeves

Table 7: Results

In all, AUTOWRAP extracted 27 columns of data from the three sites. In one site, the algorithms made a mistake in the list template, resulting in it improperly concatenating every two rows, thereby producing a table with 8 columns, rather than 4. The problem of correct list segmentation is addressed

in a different work [24], which shows how to improve AutoWrap's results with additional information from detail pages. In this work we are concerned with labeling columns rather than extracting structured records.

Nineteen of the 27 columns were correctly labeled, 9 were incorrectly labeled, and two were unlabeled, resulting in precision and recall of $P = 0.64$ and $R = 0.89$.[7] However, five of the columns that were incorrectly labeled were not in the schema we created: e.g., , "Transmission", "Stock Number", "Warranty" and "Dealer" in Table 7. When these are excluded from the incorrectly labeled columns count, precision increases to $P = 0.80$.

5 Automatic Linking

Record linkage is a process which attempts to link records from various data sources. Example application domains of record linkage include linking records from online data sources with known entities for security related applications or data warehouse applications. However, the data collected from various sources often contains errors, such as spelling mistakes, use of inconsistent acronyms, missing fields, or many others. Moreover, the collected data may not have enough common information to accurately link records across the two sources.

Over the years significant work has been done in the area of record linkage [2, 5, 8]. A key issue that arises when performing record linkage is that one of the two datasets may have additional attributes that cannot be compared to any attributes in the other dataset. Consider the following scenario: one dataset contains names and ticker symbols of various companies and the other dataset consists of company names, people associated with certain companies, city, and state information extracted from various news articles. The only common attribute between the two datasets is the company name and this is not be enough to link records together. Similar scenarios often occur in data warehousing when linking order information with products. Often the only common attributes between datasets are item names and sometimes descriptions. However, when people try to link two records they often utilize additional information to determine if two records should be linked. In our record linkage system, we incorporate this idea by acquiring supplementary information from secondary data sources.

Many record linkage systems rely on a set of pre-defined transformations which calculate some type of edit distance. This pre-defined set limits the system by confining the set of textual inconsistencies that it can resolve to the set of available transformations. Such an approach limits the robustness of a system. However, secondary sources can be exploited as specialized transformations that can provide additional resolution capabilities. Furthermore, the additional data can be used in labeling record pairs as matches or non-matches. These labeled record pairs can then be used to train the record linkage process. This second

[7] We define True Positives (TP) as correctly labeled columns; False Positive (FP) as incorrectly labeled columns; and False Negatives (FN) as unlabeled columns; therefore, $P = TP/(TP + FP)$ and $R = TP/TP + FN$.

Article Companies				
Article Source	**Company**	**Person**	**City**	**State**
Yahoo	JPC Enterprises, Inc	Joseph M. Tabak	Houston	TX
AP	Summit Bancorp	Robert G. Cox	Arlington	VA
AP	Summit Bancorp	T. Joseph Semrod	Arlington	VA
Yahoo	Lincoln Financial Group	Jon A. Boscia	Boston	MA

Forbes Companies	
Company	**Ticker**
Lincoln Electric	LECO
Summit Technology	BEAM
Summit Bank	SBGA
Summit Bancorp	SUB

Additional Information			
Company	**Person**	**City**	**State**
Lincoln Electric	J. Boscia	Boston	MA
Summit Technology	P. Wang	Houston	TX
Summit Bank	M. Shea	LA	CA
Summit Bancorp	Bob Cox	Arlington	VA

Figure 10: Examples of Records from News Articles and Companies dataset and of Additional Information for the Companies dataset

application of secondary sources can be used in automating the labeling process and minimizing user involvement in the system.

In this paper, we describe a framework for automatically identifying secondary sources for record linkage and and we show how this information can be used to improve accuracy of the record linkage process and reduce the amount of labeled data required. We begin by describing a motivating example that we use throughout the paper. Next, we provide a brief overview of the Active Atlas record linkage system and the Prometheus information mediator, both of which our system Apollo builds upon. Then, we describe how we can use information from secondary sources to improve the accuracy of the record linkage process with manually labeled data. We then describe how secondary sources can be used to automatically label the data from a very small set of initial labeled examples. Finally, we describe related work and conclude the paper by discussing ideas for future work.

5.1 Motivating Example

With the abundance of publicly available information there exists the possibility of building large information integration applications. These applications could

serve as useful tools in doing research on individuals, products, corporations, or any other entities for which public information is readily available. However, for such applications to be useful, two requirements need to be met: easy queriability of relevant data sources, and precise linkage of records across these sources. An application which doesn't meet these requirements can be useless if there is a lack of relevant information presented about a given entity, inaccurate if the information presented is falsely attributed to a given entity, or both.

Consider the example where we would like to learn as much as we can about a certain company. Imagine an information integration application which attempts to combine information to provide a detailed report about a given company. For simplicity we assume that the linkage component of this application has access to a dataset of companies. For each company, the dataset provides the company name and the ticker symbol. In addition, there exists a source which provides information extracted from various news articles. The task of the application would be to link news articles to various companies.

A problem encountered is that various news articles may refer to the same company using different names. Moreover, the news articles do not provide the ticker symbol for the company. They only provide the source of the article, company name(s), associated people (if any), and the location of the article. Figure 10 shows example records extracted from the articles and companies datasets. It should be clear that given just the names of the companies it would be hard for any linkage system to link the records from these two datasets.

In this example, record linkage plays a vital role in the effectiveness of the application. Without an accurate linkage component, the users of the application could be misinformed about the company they are researching. Therefore, it is essential that the linkage component be accurate in matching entities across the two data sources in question. Furthermore, since the role of such an application is to facilitate the research process, a user's involvement should be minimized. This means that any learning that the system needs to do should be as automated as possible while maintaining a very high level of accuracy.

5.2 Active Atlas Overview

Active Atlas' architecture consists of two separate components: a candidate generator and a mapping rule learner. Its goal is to find common entities amongst two record sets from the same domain. The candidate generator proposes a set of potential matches based on the transformations available to the system. The transformation may be one of a number of string comparison types such as equality, substring, prefix, suffix, stemming, or others and are weighted equally when computing similarity scores for potential matches. Once the candidate generator has finished proposing potential matches, Active Atlas moves on to the second stage and uses the potential matches as the basis for learning mapping rules and transformation weights.

The mapping rule learner establishes which of the potential matches are correct by adapting the mapping rules and transformation weights to the specific domain. Due to the fact that the initial similarity scores are very inaccurate, the

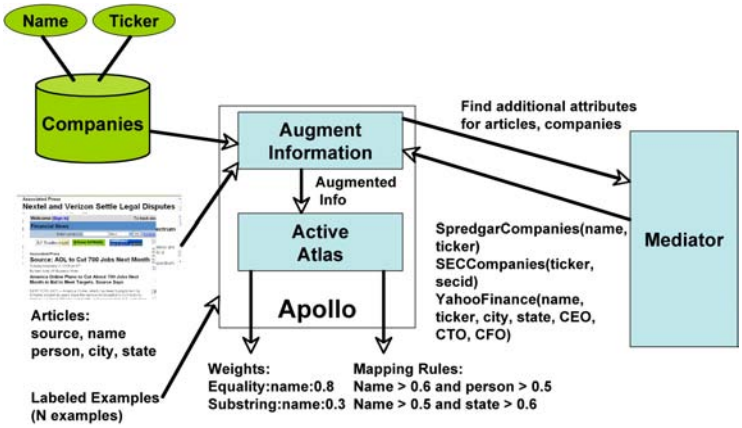

Figure 11: Architectural overview of Apollo

system uses an active learning approach to refine and improve the transformation weights and mapping rules. This approach uses a decision tree committee model for learning with N members in the committee. The mapping rule learner selects the most informative potential match and asks the user to label this example as either a match or non-match. The user's response is used to refine and recalculate the transformation weights, learn new mapping rules, and reclassify record pairs. This process continues until: (1) the committee learners converge and agree on one decision tree, or (2) the user has been asked to label a pre-defined number record pairs. Once the mapping rules and transformation weights have been learned, Active Atlas uses them to classify all the potential matches in the system as matched or not matched. The results are then made available to the user.

5.3 Prometheus Mediator

Prometheus is a mediator system [40] that builds on the inverse-rules algorithm of Duschka [14] and supports the automatic composition of web services. The system has a world model described in terms of a set of domain relations. The world model for the example application described Section 5.1 has three domain relations: *Article*, *Company*, and *Employee*. The available data sources are viewed as relations. The example application described in Section 5.1 has access to the following data sources:

ForbesArticles, *ForbesCompanies*, *SpredgarCompanies*, *SECCompanies*, and *YahooFinance*. Prometheus requires information about the semantic types of different attributes provided by each data source. The source relations with the attribute types are shown below.

ForbesArticles(source:url, company:companyname,
person:personname, city:city, state:state)

ForbesCompanies(name:companyname, ticker:ticker)
SpredgarCompanies(name:companyname, ticker:ticker)
SECCompanies(ticker:ticker, secid:secid)
YahooFinance(name:companyname, ticker:ticker, city:city,
state:state, CEO:personname, CTO:personname,
CFO:personname)

For each attribute the type information follows the attribute name. For example, the type of the *company* attribute of the *ForbesArticles* is *companyname*. Each of the available data sources are then described in terms of the domain relations as shown below.

R1:ForbesArticles(source, company, person, city, state):-
Article(source, author, city, state, company, person)
R2:ForbesCompanies(name, ticker):-
Company(name, ticker, address, city, state, secid)
R3:SpredgarCompanies(name, ticker):-
Company(name, ticker, address, city, state, secid)
R5:SECCompanies(ticker, secid):-
Company(name, ticker, address, city, state, secid)
R6:YahooFinance(name, ticker, city, state, CEO,
CTO, CFO):-
Company(name, ticker, address, city, state, secid),
Employee(name, CEO),
Employee(name, CTO),
Employee(name, CFO)

The user can pose queries to Prometheus using the domain relations. The Prometheus mediator utilizes these descriptions and attribute type information to reformulate the user queries into a set of source queries, execute the source queries, and return the results.

5.4 Automatically Augmenting Primary Data Sources

Current record linkage systems [5, 8, 11, 13, 18, 19, 28, 37, 39] excel at learning how to weigh attributes and link records across data sources. Using machine learning techniques such as decision trees [34], bagging and boosting [1, 7], or Bayesian Networks [17], they are able to determine which attributes are most relevant to consider when trying to match records across different data sources. Apollo exploits this process by automatically augmenting data sources with additional attribute(s). The machine learning component of Apollo is then able to use these attributes in learning the correct mapping rules used to classify record pairs. Using labeled examples provided by a user, the system is able to

learn if the newly added attributes are informative enough to incorporate into the mapping rules.

Figure 11 shows the architecture of the Apollo linkage system that automatically incorporates additional attributes. The core of the Apollo system is divided into two parts: (1) The information augmentation component is responsible for accepting the schemas of the two datasets from the user, querying the Prometheus mediator for available additional attributes, determining the mapping of attributes between two datasets, and retrieving values of any necessary additional attributes. (2) The Active Atlas component includes the candidate generator and mapping rule learner as described in section 5.2.

To more clearly illustrate the process of augmenting information consider the example shown in Figure 11. The user has assigned Apollo the task of linking various news articles from the *ForbesArticles* data source to companies from the *ForbesCompanies* data source. However, the only common attribute among the two datasets is the *companyname*. Apollo sends a request to Prometheus to obtain data sources that may have additional information about the *ForbesArticles* and *ForbesCompanies* sources.

Upon receiving a request from Apollo, Prometheus finds the source descriptions for both data sources. From these descriptions (rules **R1** and **R2** as shown in Section 5.3), it find all domain relations present in the body of the description. For the *ForbesArticles* and *ForbesCompanies* data sources, it finds the *Article* and *Company* domain relations respectively. Next, Prometheus looks through all its source descriptions to find the descriptions containing the selected domain relations. In our example, it finds only the *ForbesArticles* data source for the *Article* relation while, for the *Company* relation, finding the *ForbesCompanies, SpredgarCompanies, SECCompanies*, and *YahooFinance* data sources. Finally, the two primary sources are removed from the sources found in the previous step and Prometheus returns the names and attribute lists for each of the remaining data sources as available secondary data sources. In our example, no secondary data sources are found for *ForbesArticles* and *SpredgarCompanies, SECCompanies*, and *YahooFinance* data sources are returned for *ForbesCompanies*.

Apollo examines the attribute list and attribute types of the returned sources to determine which sources can be used to augment the primary data source with additional information. Each qualifying secondary data source must meet two conditions. First, it must provide at least one additional attribute to the primary source and second, this additional attribute(s) should be comparable to at least one attribute in the other primary data source. In the given example, Apollo discards the *SpredgarCompanies* data source as it only provides *name* and *ticker*, and both attribute types are already present in the *ForbesCompanies* dataset. Apollo also discards the *SECCompanies* data source as it provides an additional attribute called *secid*, but the *ForbesArticles* dataset does not contain any attribute of the same type as *secid*.

Finally, Apollo determines that the *YahooFinance* data source can be a useful secondary data source as it provides the following five additional attributes:*city, state, CEO, CTO*, and *CFO*. Moreover, the *city* and *state* attributes have the same attribute types as the *city* and *state* attributes of the

ForbesArticles relation and the *CEO, CTO*, and *CFO* attributes have same type as the *person* attribute of the *ForbesArticles* relation. Next, it queries the Prometheus mediator to obtain relevant records from the secondary data sources. In our example, it would request records of the relevant companies from the *YahooFinance* data source.

Once Apollo retrieves this additional information, it uses the candidate generator and the mapping rule learner from the Active Atlas system to determine if two records should be linked. The key difference is that these components can now learn mapping rules based on four common attributes. Originally, there existed only one common attribute between the two datasets. But as shown in the *Additional Information* table in Figure 10, Apollo can now learn on three additional attributes. These additional attributes facilitate the learning of more informative mapping rules, which leads to more precise linkage between the two datasets. This is supported by the results we present in Section 5.4.1.

5.4.1 Experimental Evaluation

To support our hypothesis that augmenting primary sources with additional information will improve the accuracy of the record linkage process while decreasing the number of required labeled examples, we performed experiments in two real world domains, restaurants and companies. In the restaurant domain, we used wrapper technology discussed in [32] to extract restaurant records from the Zagat's and Dinesite web sources. Each web source provided a restaurant's name, address, city, state, phone number, and cuisine type. Due to the inconsistencies between the two sources, a record linkage system is required to find common restaurants. Furthermore, we used the geocoder web service as a secondary source. This source takes in an address as input and outputs the corresponding latitude and longitude coordinates. The Zagat data source contained 462 records, while the Dinesite data source contained 678 records. There were 140 matching records between the two datasets.

As a first step we ran the candidate generator to obtain 4620 candidate matches. From the candidate matches, we randomly selected 50, 100, 150, 200, and 250 record pairs, labeled them as matches or non-matches, and used this as input into the Apollo[8] and Atlas[9] systems. The goal of the experiments was to show that by utilizing the geocoder secondary source, the system was able to more accurately link records across the two primary data sources using fewer labeled examples.

As shown in Figures 12 and 13, the addition of a secondary source led to a significant improvement in precision and recall. With the use of a secondary source, Apollo reached 100% precision and 97.22% recall with only 100 total labeled examples. Out of the 100 labeled examples, there were, on average, only eight positive examples. In the case of 50 labeled examples, there were only 2 or 3 positive examples. Therefore, there was not enough information (from positive

[8]Since we only had the passive learning implementation of Active Atlas available, in our experiments we only used Apollo with passive learning.

[9]Atlas is a version of the Active Atlas system which uses passive learning.

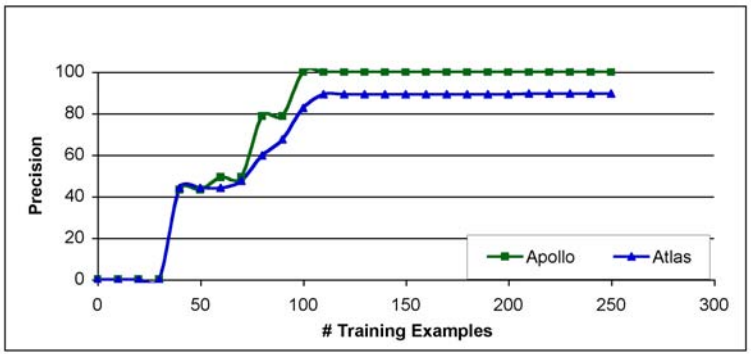

Figure 12: Precision Graph for Restaurant Domain

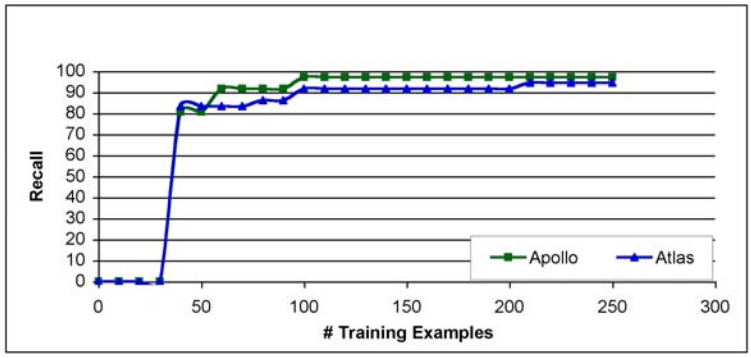

Figure 13: Recall Graph for Restaurant Domain

labeled examples) for the decision tree learner to utilize information from the secondary source. Without the secondary source, precision and recall levels were lower, even with 250 total labeled examples, then the levels seen in Apollo with just 100 total labeled examples. The improvement brought about by the secondary source is due to the secondary source's ability to handle inconsistencies for the given attribute(address) better than string transformations.

In the company domain, we extracted company records from two primary data sources: (1) A company database containing company name and ticker symbol for 503 companies, and (2) using wrapper technology discussed in [32] to extract company information containing company name, person name, and position from various news articles for 113 companies. The key challenge with the company domain data was that company name was the only common attribute and companies were referred to using different names in different articles. We utilized Yahoo Finance[10] as a secondary source and it provided the company name, ticker symbol, and three top officials for a given company.

[10]http://finance.yahoo.com

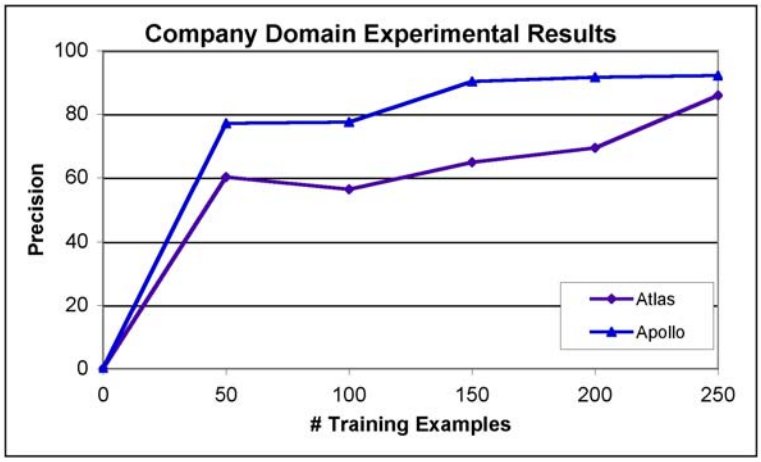

Figure 14: Precision Graph for Company Domain

As shown in Figures 14 and 15, record linkage using a secondary source achieves better precision and recall values as it is able to utilize person and company names in the linkage process. It is worth noting that company articles may mention people other than the top three officials, in which case our secondary source is less useful. This explains the lower initial recall level in Apollo. However, once enough positive training examples are provided, Apollo outperforms Atlas.

5.4.2 Utilizing Secondary Sources For Automatic Labeling

Previous work done in record linkage has assumed that a large number of training examples are available for learning. However, in large scale record linkage applications, a user is unlikely and unwilling to label a large set of examples for all possible combinations of primary sources. In this section, we introduce our approach to addressing this issue by utilizing reliable secondary sources.

There exist a wide variety of potentially useful secondary sources. The information provided by some of these sources can be used to identify matched records. An example of such a source is the geocoder. It provides sufficiently distinct information (latitude and longitude coordinates) which can determine if two addresses are the same. Sources that provide distinguishing information can therefore be used to automatically label records during the learning process to minimize the user involvement.

Intuitively, automatic labeling of record pairs using secondary sources is a two-step process. The first step consists of an evaluation of the available secondary sources to determine their viability. The second step is to use the viable secondary source(s) to automatically label record pairs to assist in the learning process. Figure 16 shows the extended architecture supporting automatic labeling and Figure 17 shows Apollo's algorithm for this two-step process. In Section

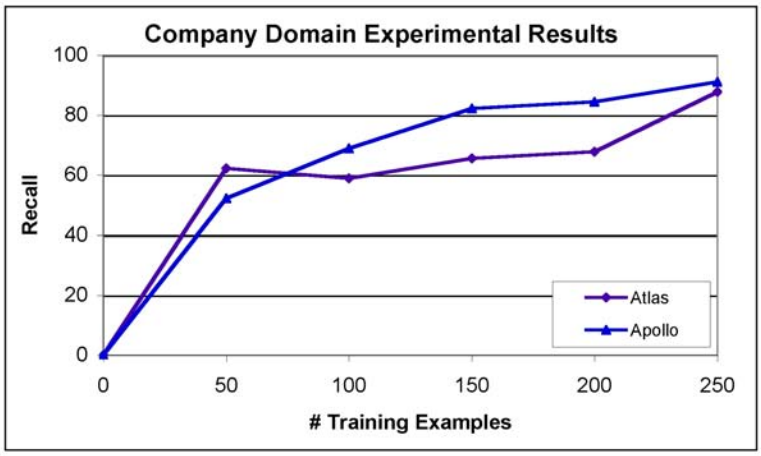

Figure 15: Recall Graph for Company Domain

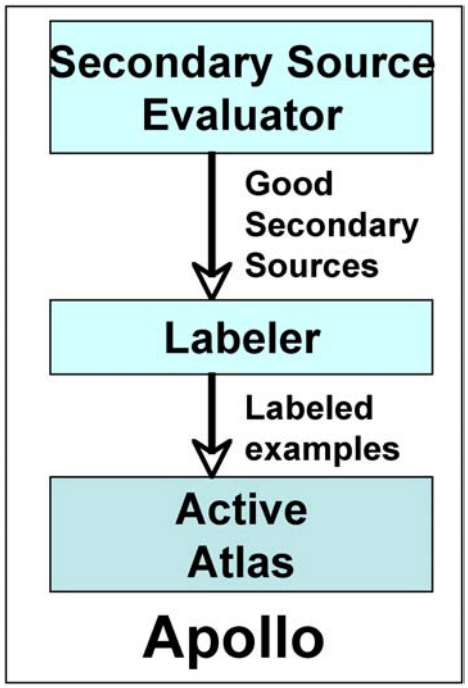

Figure 16: Architectural overview of Apollo with automatic labeling

Algorithm : ApolloLearner($LE, SS, N, AllCM$)

procedure EvaluateSS(LE, SS)
 $GoodSS \leftarrow \phi$
 for each $src \in SS$
  do { $\#true \leftarrow 0$
  $\#false \leftarrow 0$
  for each $example \in LE$
  do { $label \leftarrow$ Label($src, example$)
  if $label$
  then $\#true \leftarrow \#true + 1$
  else $\#false \leftarrow \#false + 1$
  if $(\#true \div (\#true + \#false)) > 0.75$
  then $GoodSS \leftarrow GoodSS \bigcup src$
 return ($GoodSS$)

procedure Learn($GoodSS, N, AllCM, TrainRecs$)
 for each $R \in TrainRecs$
  do { $R.ActualLabel \leftarrow$ LABEL($GoodSS, R$)
 $nTRsets \leftarrow$ DivideNSets($TrainRecs, N$)
 for each $set \in nTRsets$
  do { $dt \leftarrow$ LearnDT(set)
  for each $CM \in AllCM$
  do { $label[CM, dt] \leftarrow$ Classify(dt, CM)
 $nextExp \leftarrow$ GetMostInformativeExp($label$)
 if $nextExp$ **not** ϕ
  then { $TrainRecs \leftarrow TrainRecs \bigcup nextExp$
  Learn($GoodSS, N, AllCM, TrainRecs$)
 return (ReturnMatches($label$))

Figure 17: Apollo's Unsupervised Learning Algorithm

5.4.3 we describe our approach to evaluating the available secondary sources, and in Section 5.4.4 we describe Apollo's automatic labeling process.

5.4.3 Evaluating Secondary Sources

In Apollo, we provide a simple mechanism to evaluate the capabilities of various secondary sources with respect to labeling matched records. As the first step in the evaluation process, Apollo uses the candidate generator to construct a a very small set of record pairs (e.g. 25), and then asks the user to label these record pairs. These labeled pairs (LE) and the set of available secondary sources (SS) are then passed to the *EvaluateSS* procedure shown in Figure 17.

Next, for each secondary source, Apollo uses the $LABEL$ method to classify all of the record pairs as matches or non-matches. Apollo compares the user provided labels with the labels generated using each secondary source to obtain the percentage of record pairs that are labeled correctly. If the secondary source can classify more then 75%[11] of the record pairs correctly, we classify it as being useful for labeling. Using this approach, a user's involvement is limited to this step and only requires a user to label a very small set of record pairs (25 in our implementation).

5.4.4 Labeling Training Examples

Once Apollo has identified which secondary source(s) can be used to automatically label record pairs, it uses this source to generate a set of labeled training examples. It utilizes an active learning approach described by Tejada et al. [39] to reduce the number of examples labeled by the secondary source. This reduction is useful when the available sources are either slow or fee-based.

Before the labeling process can begin, Apollo randomly selects 25 record pairs as initial training data to start decision tree learning. As shown in the *Learn* procedure in Figure 17, the unsupervised learning process accepts: a set of viable secondary sources ($GoodSS$), the number of members in the decision tree committee$N$), the set of candidate matches($AllCM$), and a set of randomly selected training records($TrainRecs$). Apollo labels each record pair R in the training set using the selected secondary source(s). In the traditional record linkage process, this step would require a user to provide these labels. However, Apollo's use of secondary sources relieves the user of this burden.

Next, Apollo divides the record pairs in the training set into N unique sets, and learns a decision tree(dt) based on each set. The learned decision tree(s) is then used to classify each record pair(CM) in the set of candidate matches($AllCM$). It then determines the most informative examples by choosing record pairs from the candidate set, with the highest level of disagreement between the N decision trees. Finally, if all of the decision trees converge (there are no informative examples found), Apollo returns all record pairs classified as matches to the user(the $RETURNMATCHES$ method examines the labels

[11]This is a manually selected value and we are working on a method to learn this value automatically.

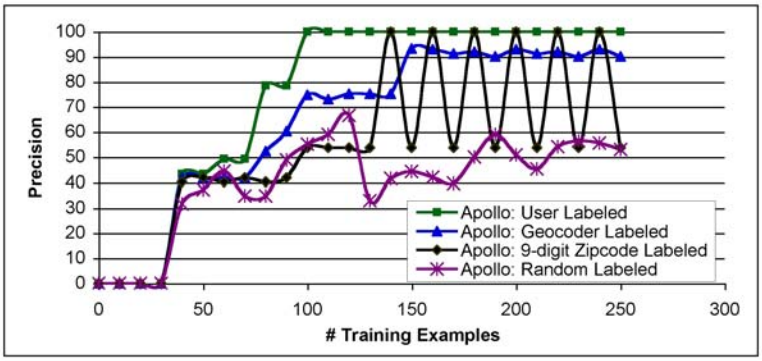

Figure 18: Precision Graph for Restaurant Domain with Automatic Labeling

assigned to all the record pairs and returns only the matches). Otherwise, the *LEARN* process is repeated with all of the informative examples added to the training set.

Experimental Evaluation We evaluated the idea of utilizing secondary sources to automatically label record pairs as matches or non-matches by performing four sets of experiments in the restaurant domain and one set of experiments in the company domain. The datasets for each domain are explained in Section 5.4.1. The goal of the experiments was to show that we could maintain a high level of accuracy with minimal user involvement. User involvement was limited to providing 25 labels used for the evaluation of the available secondary sources.

In the first four sets of experiments we performed linkage across datasets in the restaurant domain. Available secondary data sources included: a geocoder that provided geographic coordinates for a given address, and a postal web site which provided 9-digit zip code for a given address. Each set of experiments contained 10 runs and the results shown are the average values for all runs.

In the first set of experiments we utilized the geocoder to generate training examples. If the geocoder returned the same geographic coordinates for the address of the two records in a record pair, the record pair was labeled as a match. In the subsequent sets, we performed the experiments using the postal service web site, a random method, and manual user labeling. In the random method, we randomly generated labels for the training examples while in the manual labeling method, a user was responsible for labeling all of the training examples (up to 250). We include the random method as a baseline comparison. The performance of the Apollo system with each secondary source was compared to the performance of the Apollo system requiring a user to label record pairs and to random labeling.

Figures 18 and 19 show that the Apollo system with user labeling performs slightly better when compared to the Apollo system with secondary source labeling. In particular, the Apollo system with the geocoder secondary source

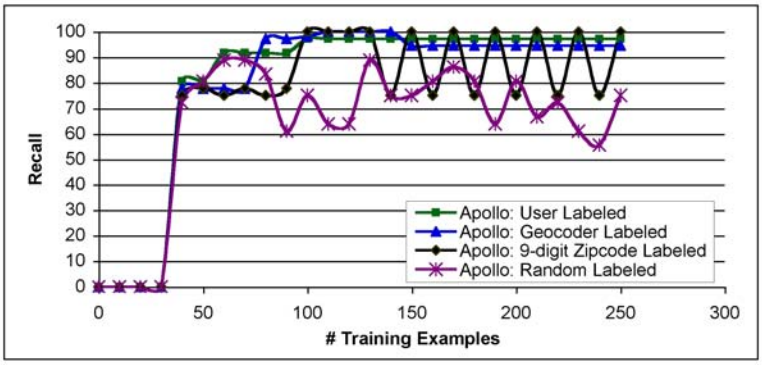

Figure 19: Recall Graph for Restaurant Domain with Automatic Labeling

reaches an optimal performance of 94.44% precision and 92.86% recall with 150 labeled examples. The Apollo system with user labeling reaches an optimal performance of 100% precision and 97.22% recall with 100 labeled examples. However, in the case of the Apollo system with the user labeling, the user has the task of labeling 100 record pairs manually, while in the Apollo system with automatic unsupervised labeling, the user only needs to label 25 record pairs used in evaluating the secondary sources.

The Apollo system with the postal service web site also gets to 100% precision and 75% recall with 140 labeled examples. Due to the fact that there exist different restaurants with the same 9-digit zip code, the system alternates between learning strict and looser mapping rules. Because of inaccurate labeling, an oscillation between 100% precision and 100% recall occurs. This oscillating effect can be seen in the zip code results in Figures 18 and 19. Finally, the Apollo system with either secondary sources outperforms the "straw man" approach of randomly assigning labels to the record pairs, as expected.

In the company domain, we used the officer information provided by Yahoo Finance secondary source to automatically label record pairs for learning. We compared this with a user labeling method, with and without secondary sources. As shown in Figure 20, Apollo with automatic labeling achieves higher precision. However, as seen in Figure 21, it has lower recall values. This is expected because some articles mention people other then the top three officers as returned by Yahoo Finance. For example, not all articles about large companies such as Microsoft would mention one of the top three officials such as Bill Gates, instead mentioning a public relations representative. Therefore, the secondary source incorrectly labels some true positives as negatives.

Even with this introduction of noise (calculated to be roughly 20% in our experiments), Apollo with automatic labeling still managed to reach the highest level of precision while maintaining a level of recall which on average was only 10% lower then the other methods. As in the restaurant domain, the user was only required to label 25 record pairs, much less then the 250 required to be

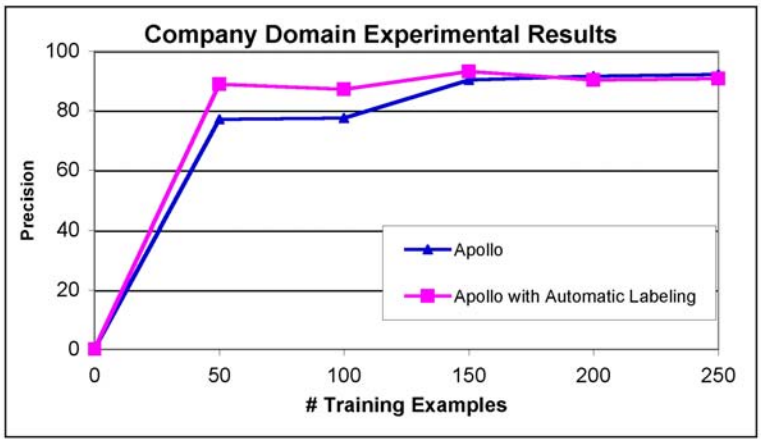

Figure 20: Precision Graph for Company Domain with Automatic Labeling

labeled in the other methods.

While Apollo with automatic labeling using secondary sources perform well in both domains, it performs slightly worse than the Apollo system with user labeled examples. The key reason behind this is due to the inaccurate labeling mentioned earlier. In general, it would be difficult to find one "golden" secondary source that can accurately identify records as matches or non-matches. We could improve the accuracy of the labeling process by combining information from multiple secondary sources, or by combining secondary source information with attributes of the primary sources to label record pairs.

6 Transition Efforts

During the course of the contract, we received money from the project sponsor to transition the technology for extracting data from semi-structured (HTML-based) sites and apply it to semi-structured text for the purposes of extracting data from a more broad category of sources. This technical effort was successful, and led the project sponsor to consider an application of the technology for harvesting data from a specific category of web sites of interest to them. Based on our analysis of this application, the project sponsor has funded Fetch Technologies, in a separate contract, to build the prototype of this application, which Fetch is currently doing. This prototype involves extending the WideLink automatic wrapper learning technology to HTML sites that change, and specifically targets websites in the category noted earlier.

We are involved in two other transition efforts. In the first effort, Fetch is extending the automatic navigation and AutoWrap technology for DARPA's Calo program. This technology enables Calo users to easily create information agents. Fetch is also using the technology commercially with customers includ-

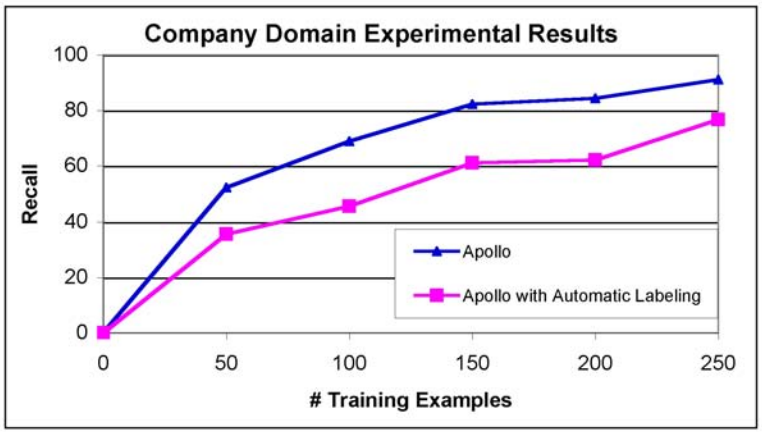

Figure 21: Recall Graph for Company Domain with Automatic Labeling

ing Sony, JD Power, BizRate, as well as several other customers in background search industry.

7 Conclusion and Future Directions

Central to our approach is the idea that online sources contain much explicit and implicit structure that we can exploit for the purposes of automatic data extraction, labeling and linking. Our main technical contribution are unsupervised machine learning algorithms that take advantage of underlying structures to automate information tasks. By analyzing a set of pages within a site, our algorithms can classify them into types, such as list and detail pages, and find common structure within pages of the same type and use it to extract data from the pages. By exploiting data redundancies between list and detail pages, our algorithms can segment extracted data into distinct records or relations. Moreover, we can learn the structure of information on one site within a domain and use it to label automatically extracted data on a new site within the same domain. In addition, redundancies in information across sites can be used to link records, as well as to augment information we have about the entity to get a more complete view of it.

We showed that the semi-automatic and fully automatic methods we have developed in the course of the project can greatly speed up information integration tasks. We are continuing to improve the technology to make these methods more robust and scalable.

Supplement: WideLink Publications

The following publications were funded completely or in part by this project:

1. Greg Barish and Craig A. Knoblock. Learning value predictors for the speculative execution of information gathering plans. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-2003)*, Acapulco, Mexico, 2003.

2. Kristina Lerman, Steven Minton and Craig Knoblock, "Wrapper Maintenance: A Machine Learning Approach," J*ournal of Artificial Intelligence Research.*

3. Kristina Lerman, Lise Gettor, Steven Minton and Craig Knoblock, "Using the Structure of Web Sites for Automatic Segmentation of Tables", In *Proceedings of SIGMOD-2004*, Paris, France, 2004.

4. Kristina Lerman, Cenk Gazen, Steven Minton and Craig Knoblock, "Populating the Semantic Web," In *Proceedings of the AAAI Workshop on Automatic Text Extraction and Mining*, San Jose, CA, 2004.

5. Kristina Lerman, Steven Minton and Craig Knoblock, "Machine Learning Techniques for Web Wrapper Maintenance," invited submission to a chapter in *Virtual Enterprise Integration: Technological and Organizational Perspective*, Goran Putnik and Maria M. Cunha (Eds).

6. Martin Michalowski, Snehal Thakkar, and Craig Knoblock. Exploiting Secondary Sources for Automatic Object Consolidation. In *KDD workshop on Data Cleaning, Record Linkage and Object Consolidation*, August 2003.

7. Ion Muslea, Steven Minton, and Craig A. Knoblock. Active learning with strong and weak views: A case study on wrapper induction. In *18th International Joint Conference on Artificial Intelligence (IJCAI-2003)*, Acapulco, Mexico, 2003.

8. Ion Muslea, Steve Minton, and Craig Knoblock Active + Semi-Supervised Learning = Robust Multi-view Learning, In *Proceedings of the 19th International Conference on Machine Learning (ICML-2002*), 2002.

9. Ion Muslea, Steve Minton, and Craig Knoblock Adaptive View Validation: A First Step Towards Automatic View DEtection", In *Proceedings of the 19th International Conference on Machine Learning (ICML-2002)*, 2002.

10. Ion Muslea, Steve Minton, and Craig Knoblock Adaptive View Validation: A Case study on Wrapper Induction and Text Classification", In *Proceedings of the Workshop on Intelligent Services Integration at the National Conference on Artificial Intelligence (AAAI-2002)*, 2002.

11. Snehal Thakkar, Craig A. Knoblock, Jose Luis Ambite, and Cyrus Shahabi Dynamically Composing Web Services from On-line Sources In *Proceedings of the AAAI Workshop on Intelligent Service Integration at the National Conference on Artificial Intelligence (AAAI-2002)*, 2002.

12. Snehal Thakkar and Craig A. Knoblock.Efficient execution of recursive integration plans. In *Proceedings of IJCAI Workshop on Information Integration on the Web*, Acapulco, Mexico, 2003.

13. Steven Minton, Sorin Ticrea, Jennifer Beach. Trainability: Developing a responsive learning system, In *Proceedings of IJCAI Workshop on Information Integration on the Web*, Acapulco, Mexico, 2003.

14. Snehal Thakkar, Jose-Luis Ambite, and Craig A. Knoblock. A view integration approach to dynamic composition of web services. In *Proceedings of the ICAPS Workshop on Planning for Web Services*, Trento, Italy, 2003.

15. Snehal Thakkar and Craig A. Knoblock, Efficient Execution of Recursive Integration Plans, In *Proceeding of 2003 IJCAI Workshop on Information Integration on the Web*, August 2003.

References

[1] N. Abe and H. Mamitsuka. Query learning strategies using boosting and bagging. In *Proceedings of the Fifteenth International Conference on Machine Learning*, 1998.

[2] R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating fuzzy duplicates in data warehouses. In *Proceedings of the 28th International Conference on Very Large Databases (VLDB) 2002*, 2002.

[3] A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2003.

[4] L. Arlotta, V. Crescenzi, G. Mecca, and P. Marialdo. Automatic annotation of data extracted from large web sites. In *Proceedings of the Sixth International Workshop on Web and Databases (WebDB03)*, 2003.

[5] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2003)*, pages 39–48, Washington DC, 2003.

[6] V. Borkar, K. Deshmukh, and S. Sarawagi. Automatic segmentation of text into structured records full text. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, Special Interest Group on Management of Data, pages 175–186. ACM, 2001.

[7] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[8] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and efficient fuzzy match for online data cleaning. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, San Diego, CA, 2003. ACM Press.

[9] H. Chen, S. Tsai, and J. Tsai. Mining tables from large scale html texts. In *18th International Conference on Computational Linguistics (COLING)*, 2000.

[10] W. Cohen and L. Jensen. A structured wrapper induction system for extracting information from semi-structured documents. In *Proceedings of the IJCAI Workshop on Adaptive Text Extraction and Mining*, 2001.

[11] W. W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 201–212. ACM Press, 1998.

[12] V. Crescenzi, G. Mecca, and P. Merialdo. RoadRunner: Towards automatic data extraction from large web sites. In *Proceedings of the 27th Conference on Very Large Databases (VLDB)*, Rome, Italy, 2001.

[13] A. Doan, Y. Lu, Y. Lee, and J. Han. Object matching for data integration: A profile-based approach. In *Proceedings of the IJCAI-03 Workshop on Information Integration on the Web*, 2003.

[14] O. M. Duschka. *Query Planning and Optimization in Information Integration.* PhD thesis, Stanford University, 1997.

[15] D. Freitag and N. Kushmerick. Boosted wrapper induction. In *Proceedings of the 7th Conference on Artificial Intelligence (AAAI-2000)*, pages 577–583. AAAI Press, Menlo Park, CA, July 2000.

[16] Z. Ghahramani and M. I. Jordan. Factorial hidden Markov models. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Proc. Conf. Advances in Neural Information Processing Systems, NIPS*, volume 8, pages 472–478. MIT Press, 1995.

[17] D. Heckerman and R. Shachter. Decision-theoretic foundations for causal reasoning. *Journal of Artificial Intelligence Research*, 3:405–430, 1995.

[18] M. A. Hernndez and S. J. Stolfo. The merge/purge problem for large databases. In *Proceedings of the ACM SIGMOD Conference*, 1995.

[19] L. Jin, C. Li, and S. Mehrotra. Efficient record linkage in large data sets. In *Proceedings of the 8th International Conference on Database Systems for Advanced Applications (DASFAA 2003)*, Kyoto, Japan, 2003.

[20] N. Kushmerick and B. Thoma. *Intelligent Information Agents R&D in Europe: An AgentLink perspective*, chapter Adaptive information extraction: Core technologies for information agents. Springer, 2002.

[21] N. Kushmerick, D. S. Weld, and R. B. Doorenbos. Wrapper induction for information extraction. In *Proceedings of the Intl. Joint Conference on Artificial Intelligence (IJCAI)*, pages 729–737, 1997.

[22] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th International Conf. on Machine Learning*, pages 282–289. Morgan Kaufmann, San Francisco, CA, 2001.

[23] K. Lerman, C. Gazen, S. Minton, and C. A. Knoblock. Populating the semantic web. In *Proceedings of the Workshop on Advances in Text Extraction and Mining (AAAI-2004)*, 2004.

[24] K. Lerman, L. Getoor, S. Minton, and C. A. Knoblock. Using the Structure of Web Sites for Automatic Segmentation of Tables. In *In* Proceedings of ACM SIG on Management of Data (SIGMOD-2004), jun 2004.

[25] K. Lerman and S. Minton. Learning the common structure of data. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-2000)*, Menlo Park, July 26–30 2000. AAAI Press.

[26] K. Lerman, S. Minton, and C. Knoblock. Wrapper maintenance: A machine learning approach. *Journal of Artificial Intelligence Research*, 18:149–181, 2003.

[27] M. Michalowski, S. Thakkar, and C. A. Knoblock. Exploiting secondary sources for unsupervised record linkage. In *In Proceedings of the 2004 VLDB Workshop on Information Integration on the Web*, 2004.

[28] A. E. Monge and C. Elkan. The field matching problem: Algorithms and applications. In *Proceedings of the Second Conference on Knowledge Discovery and Data Mining*, pages 267–270, 1996.

[29] K. Murphy. Dynamic bayesian networks: Representation, inference and learning, 2002.

[30] I. Muslea, S. Minton, and C. Knoblock. Active + semi-supervised learning = robust multi-view learning. In *Proceedings of the 19th International Conference on Machine Learning (ICML 2002)*, pages 435–442. Morgan Kaufmann, San Francisco, CA, 2002.

[31] I. Muslea, S. Minton, and C. A. Knoblock. Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems*, 4:93–114, 2001.

[32] I. Muslea, S. Minton, and C. A. Knoblock. Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems*, 4(1/2), 2001.

[33] D. Pinto, A. McCallum, X. Wei, and W. B. Croft. Table extraction using conditional random fields. In *Proceedings of the ACM SIGIR)*, 2003.

[34] J. R. Quinlan. Improved use of continuous attributes in c4.5. *Journal of Artficial Intelligence Research*, 4:77–90, 1996.

[35] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Readings in Speech Recognition.*

[36] S. Raghavan and H. Garcia-Molina. Crawling the hidden web. In *Proceedings of the Twenty-seventh International Conference on Very Large Databases*, 2001.

[37] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *Proceedings of the 8th ACM SIGKDD Conference on Knowldge Discovery and Data Minning*, Edmonton, Alberta, Canada, 2002.

[38] S. Tejada, C. A. Knoblock, and S. Minton. Learning object identification rules for information integration. *Information Systems)*, 26(8), 2001.

[39] S. Tejada, C. A. Knoblock, and S. Minton. Learning domain-independent string transformation weights for high accuracy object identification. In *Proceedings of the Eighth ACM SIGKDD International Conference*, Edmonton, Alberta, Canada, 2002.

[40] S. Thakkar, J. L. Ambite, and C. A. Knoblock. A data integration approach to automatically composing and optimizing web services. In *Proceedings of 2004 ICAPS Workshop on Planning and Scheduling for Web and Grid Services*, 2004.

[41] J. P. Walser. Wsat(oip) package.

[42] J. P. Walser. *Integer Optimization by Local Search: A Domain Independent Approach*, volume 1637 of *LNCS*. Springer, New York, 1999.